
THE UNIVERSITY
*of* EDINBURGH

# Heterogeneous Processor Composition: Metrics and Methods

*Erik-Arne Tomusk*


Doctor of Philosophy

Institute for Computing Systems Architecture

School of Informatics

University of Edinburgh

2016

# Abstract

Heterogeneous processors intended for mobile devices are composed of a number of different CPU cores that enable the processor to optimize performance under strict power limits that vary over time. Design space exploration techniques can be used to discover a *candidate set* of potential cores that could be implemented on a heterogeneous processor. However, candidate sets contain far more cores than can feasibly be implemented. Heterogeneous processor composition therefore requires solutions to the *selection problem* and the *evaluation problem*. Cores must be selected from the candidate set, and these cores must be shown to be quantitatively superior to alternative selections. The qualitative criterion for a selection of cores is *diversity*. A diverse set of heterogeneous cores allows a processor to execute tasks with varying dynamic behaviors at a range of power and performance levels that are appropriate for conditions during runtime.

This thesis presents a detailed description of the selection and evaluation problems, and establishes a theoretical framework for reasoning about the runtime behavior of power-limited, heterogeneous processors. The evaluation problem is specifically concerned with evaluating the collective attributes of selections of cores rather than evaluating the features of individual cores. A suite of metrics is defined to address the evaluation problem. The metrics quantify considerations that could otherwise only be evaluated subjectively. The selection problem is addressed with an iterative, diversity-preserving algorithm that emphasizes the flexibility available to programs at runtime. The algorithm includes facilities for guiding the selection process with information from an expert, when available. Three variations on the selection algorithm are defined.

A thorough analysis of the proposed selection algorithm is presented using data from a large-scale simulation involving 33 benchmarks and 3000 core types. The three variations of the algorithm are compared to each other and to current, state-of-the-art selection techniques. The analysis serves as both an evaluation of the proposed algorithm as well as a case study of the metrics.

# Lay Summary

The technological developments that have led to the heterogeneous processor composition problem share many similarities with the growth of a small business. A small business might start with only one or two people, and each person must perform a broad range of tasks. The same person might work on a product on Monday, file taxes on Tuesday, fix IT issues on Wednesday, write a press release on Thursday, and mop the floor on Friday. As the business grows, people with more specialized skills are hired. There are two reasons for this: First, it is not cost-effective for a business to employ many people who have the broad skill sets of the business's founders. Such versatile people expect high salaries in line with their abilities, but most of their abilities will be wasted as they focus on a small set of tasks. Second, people with narrow, specialized skill sets are likely to be more efficient in their areas of expertise. An accountant might only need to come in one day a week, for example.

A similar specialization over time can be seen in processors. A processor handles all the calculations and logic operations in a computer. Up until the early 2000s, the majority of processors contained a single, general-purpose "worker," or CPU core. Technological advances had been increasing the performance of the core, but there had always been one core. As further increases to performance became more difficult to achieve, processor designers began to focus on increasing the number of cores on a processor instead of increasing the performance cores. This resulted in dual-core processors, and eventually quad-core and six-core processors. The first wave of *multicore* processors were *homogeneous*—all cores were identical. To extend the above analogy, this is comparable to a small group of entrepreneurs working together. As above, there are two problems with increasing the number of identical cores on a processor: First, high-performance cores are very expensive in terms of their power consumption. A large fraction of the processor market is made up of mobile processors intended for smartphones, where power is at a premium. Second, it is often the case that there is not enough work for all the cores. If all cores are identical, some computational resource is wasted. These considerations have led researches to study *heterogeneous* processors, where cores are specialized to different types of tasks and are used only when required.

Selecting cores for a heterogeneous processor and selecting employees for a business have common questions that must be answered: How much work will each worker perform? Is it better to have a few workers with many duties, or many workers with

a few duties? How good is the division of labor, and can it be improved? This thesis presents a description of these problems for power-limited, mobile processors. It then contributes a suite of metrics for evaluating selections of heterogeneous CPU cores, and an algorithm for selecting cores. The metrics and selection algorithm focus on providing a processor with the flexibility to maximize performance for a range of different tasks even as operating conditions change.

# Acknowledgements

I would first like to thank my academic advisers, Prof. Michael O'Boyle and Dr. Christophe Dubach, for their advice and insight over the past four years. A thank you also goes to the members of the CArD group for the discussions, and for the chit-chat.

To all the people in Edinburgh and beyond who have shared their time, shared their friendship, opened their homes, shared a meal: The things that might seem small are the things that make the biggest difference.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

*Erik-Arne Tomusk*

# Publications

Some of the contents of this thesis have appeared previously in the following publications:

- Erik Tomusk, Christophe Dubach, and Michael O'Boyle. Diversity: A design goal for heterogeneous processors. *Computer Architecture Letters*, 2015. DOI: 10.1109/LCA.2015.2499739.
  —**Chapter 5** is an extension of this publication.

- Erik Tomusk, Christophe Dubach, and Michael O'Boyle. Four metrics to evaluate heterogeneous multicores. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4), Nov. 2015. DOI: 10.1145/2829950.
  —**Chapter 6** is an extension of this publication.

- Erik Tomusk, Christophe Dubach, and Michael O'Boyle. Measuring flexibility in single-ISA heterogeneous processors. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2014. DOI: 10.1145/2628071.2628125.
  —This publication contains early versions of **section 6.6.3** and **section 6.7**.

- Erik Tomusk and Michael O'Boyle. Weak heterogeneity as a way of adapting multicores to real workloads. In *Proceedings of the 3rd International Workshop on Adaptive Self-Tuning Computing Systems (ADAPT)*. ACM, 2013. DOI: 10.1145/2484904.2484909.
  —**Section 5.6.2** is motivated by this publication.

Soli Deo Gloria

# Table of Contents

# Chapter 1:

# Introduction

Advances in semiconductor fabrication technology are continuing to increase the amount of logic that can be implemented on a chip. The almost inevitable consequence of this trend has been the proliferation of *multicore* processors. There are diminishing returns to ever larger processor cores, and it is much easier to make effective use of the additional logic by increasing the number of processing cores than by increasing the size of a single core. However, there are also diminishing returns to increasing the number of cores on a processor. This is partly because there are few computational problems that can effectively utilize a large number of cores, and partly because problems that can utilize many cores run up against power and thermal limits of processors. There is little reason to include more identical cores on a processor than can be used at once, but if the cores are different, then there is no need for all cores to be usable simultaneously. In a *heterogeneous* processor, different types of cores can be powered on demand, as required by software. The maximum power consumption of the processor could far exceed physical limits, but since only a fraction of the processor is powered at any given time, power budgets are not exceeded. Heterogeneous processors can use information that is only available at runtime to determine the number and types of active cores. This allows heterogeneous processors to accommodate various types of computational problems and to work under various runtime limits and requirements.

## 1.1 Problem Summary

A processor should be designed for a given *problem space*—the software that the processor must run, and the set of conditions under which the processor must operate. In an ideal world, it would be possible to tune a multicore processor to exactly the

requirements of a given problem space. Some computational problems might have very little variation, and would be best served by a *homogeneous* multicore processor that only implements one type of core. Some computational problems might have high variation, and would benefit from many different types of cores. Some processors might be subject to a broad range of external conditions, and would require the ability to run the same tasks in different ways. Some computational problems might exhibit well-defined, regular behavior, and would be able to make effective use of highly specialized hardware accelerators.

Tuning a processor would involve first determining the best set of computational resources for the given problem, and then determining the best balance of resources. In the case of a homogeneous processor, this would involve determining whether there should be many small cores, a few large cores, or some option in between. In the case of a processor with many accelerators, this would involve determining the balance between general-purpose and specialized hardware, the capabilities of each accelerator, etc. A large body of research addresses various, specific instances of the design problem, and many of the relevant works are discussed in **chapter 3**. However, a general, universally applicable, methodology for tuning processors appears to be far beyond the current capabilities of the architectural community.

Not only is it currently impossible to determine the best processor architecture for a problem space, but there is also very little agreement on what *best* even means in this context. I.e., in addition to a severe lack of rigorous design methodologies, there is also a lack of evaluation methodologies. The design problem is, in fact, dependent on the evaluation problem: Because the standards by which processors should be evaluated are unclear, it is difficult to develop design methodologies to meet those standards.

The general processor design problem is intractable with currently available techniques. This thesis simplifies the design problem in the following four ways: First, the problem is limited to power-constrained mobile devices. Heterogeneity is known to be beneficial for mobile devices, but rigorous design methodologies are still in their infancy. This is in contrast to server processors, where questions have been raised regarding the benefits of heterogeneity. Second, the problem is limited to *CPU core selection* rather than the design of an entire processor. Current research struggles to differentiate between effects from cores and effects from the rest of the processor. The focus on selection is intended to clarify the issue. Third, the problem is limited to only one instruction set architecture (ISA)—GPUs (graphics processing units) and hardware accelerators are explicitly excluded. Fourth, the problem is limited to only

determining which types of cores a processor should implement; the number of each type of core is not considered. The problem of selecting single-ISA core types must be solved before multiple ISAs and multiple instances of a core can be considered. Later chapters will show that even this small subset of the design problem is poorly understood and largely unaddressed.

## 1.2 Contributions

This thesis makes two types of contributions. The first type consists of the technical methods for evaluating and selecting cores. The technical methods are dependent on the second type of contribution—the theoretical framework for reasoning about the design problem. As described above, the multicore processor design problem is notoriously complex. A substantial part of this thesis is devoted to thoroughly describing the problem and establishing a set of assumptions that can be used to address it. In addition to enabling evaluation and selection methods, a well-defined framework for addressing the problem improves the interpretability of results and motivates future work. Clear assumptions aid in determining the significance of results. Future work can modify the known assumptions and extend the technical methods to other problems.

The primary contributions of the thesis are as follows:

1. Core selection is identified as a problem independent from design space exploration. As described in the previous section, the selection problem is reduced to a manageable size. Current research often attempts to solve versions of the design problem that are too large, leading to results that are difficult to interpret and generalize.

2. A novel model is proposed for reasoning about the runtime behavior of heterogeneous processors. The model is based on a probabilistically determined available runtime power distribution, and enables analyses that were previously impossible.

3. Cores are selected and evaluated based on measurable features rather than implementation details. This ensures that the proposed methods are portable to different problem spaces.

4. A set of eight metrics are defined for evaluating heterogeneous selections of cores under probabilistically varying power budgets.

5. An algorithm is defined for selecting cores to maximize the runtime flexibility of a heterogeneous processor for a given set of tasks under a probabilistically varying power budget.

## 1.3 Thesis Structure

The structure of the thesis is as follows:

**Chapter 2** presents a basic overview of processor evaluation and architecture concepts, with a focus on the terminology used throughout the thesis.

**Chapter 3** builds on the background information in **chapter 2** by reviewing related work and the current state of the art in processor architecture, design, modeling, and evaluation.

**Chapter 4** describes the infrastructure used to generate an example dataset. The example dataset is based on the microarchitectural considerations and software tools introduced in the two preceding chapters, and forms the basis for the analyses in later chapters.

**Chapter 5** provides a thorough description of the heterogeneous processor composition and evaluation problems. It is shown that the current state of the art does not adequately address the requirement for runtime flexibility in power-constrained, mobile devices. The chapter defines the difference between *design space exploration* and *core selection*, and establishes a theoretical framework for reasoning about the runtime behavior of selected cores.

**Chapter 6** defines eight new metrics for evaluating selections of heterogeneous cores. The metrics are designed for power-limited mobile devices. Each metric is illustrated with examples drawn from the example dataset, and is accompanied by a discussion of its potential uses.

**Chapter 7** presents the *LUCIE* algorithm for heterogeneous core selection. LUCIE selects cores by iteratively removing the least useful core from a candidate set. Three variations of the LUCIE algorithm are defined.

**Chapter 8** compares the LUCIE algorithm to state-of-the-art selection algorithms using the metrics defined in **chapter 6**. Selection is carried out on the example dataset. LUCIE is shown to be more appropriate for mobile devices than current techniques. In addition to evaluating LUCIE, this chapter serves as a case study of the application of the metrics.

**Chapter 9** concludes the thesis. A critical analysis of results is presented, and avenues for future work are discussed.

**Appendix A** contains the microarchitectural configurations of the processor cores in the example dataset.

## 1.4 Acknowledgements

The following people are acknowledged for their contributions to specific sections of the thesis. Peter Henderson made his collection of CPU utilization data available for **section 8.11** (p. 165). The anonymous reviewers of PACT 2014 suggested ANOVA analysis, which has been included in **section 4.3.3** (p. 44). The anonymous reviewers of SIGMETRICS 2015 recommended the KS test as a statistical technique similar in intent to the localized non-uniformity metric. The two are discussed in **section 6.6** (p. 81).

# Chapter 2:

# Background

This chapter presents fundamental concepts and terminology required for understanding processor evaluation techniques and processor architectures. Later chapters assume this background knowledge when addressing the problems of heterogeneous processor composition and evaluation.

## 2.1 Introduction

Composing a heterogeneous processor from a collection of CPU cores requires methods for selecting cores and metrics for evaluating selections. This chapter is concerned with summarizing basic background information and defining a consistent set of terms for later chapters to use when addressing processor evaluation and design; the next chapter reviews the literature in these areas. Background information is presented on the two themes of this thesis: evaluation of hardware and hardware design. The section on evaluation techniques deliberately precedes the section on hardware. While it may seem intuitive to first design hardware and then evaluate it, the reality is that the design process must be informed by the evaluation goals. Targets must exist before a processor can be designed to meet those targets.

Modern processors are designed by large groups of people. The following sections and the remainder of this thesis use *designer* as shorthand for the design team and its EDA (electronic design automation) tools.

## 2.2 Evaluation

A rigorous evaluation of a processor always requires *metrics*—quantitative figures of merit that can be analyzed. Once one or more metrics are chosen, the processor can be

optimized for those metrics. There is a potentially infinite number of processor features that could be quantified, and the choice of metrics used is therefore fundamental to the design of a processor. Any design process involves trade-offs, where improving one metric deteriorates some other metric. If the metrics that deteriorate are unimportant for the particular processor, then this is not an issue. The problem arises if important metrics are not evaluated, either due to an oversight, or due to a misunderstanding of design requirements and metrics. For example, a designer could invest substantial effort into optimizing a processor for a given metric, only to realize that the processor must perform well at a subtly different metric. Consequently, great care must be taken to ensure that metrics are understood, and that they match design requirements.

The ultimate goal of a metric is to understand the complex behavior of a processor in simple terms. There is, however, a perverse trade-off between understanding a metric and understanding a processor. A simple metric can be easy to apply and can have an obvious interpretation, but it may completely fail to provide insight into the behavior of a processor. A complex, abstract metric might accurately capture a fine detail of a processor, but the metric's overall significance might be difficult to determine. A complex metric could also hide important details (either on accident or intentionally). The following three sections discuss common metrics in order from the obvious to the abstract. The metrics are classified into the somewhat subjective categories of physical quantities, rates, and summary metrics. A fourth section discusses the benchmarking aspect of evaluation.

### 2.2.1 Physical Quantities

The most basic metrics are physical, measurable quantities. Execution *time*—how long a task takes to complete—has historically been the most important metric. Time is sometimes also called *delay*, and is usually measured in seconds. Executing a task is not without cost. The most obvious cost is the required *energy*, usually measured in joules (J) or kilowatt-hours (kWh). Energy translates directly into a monetary cost, as the electrical energy used for execution must be paid for. Energy is particularly important for mobile devices, as batteries store a limited amount. Total energy consumption is important, but so is instantaneous energy, or *power*. Power is measured in Watts (W). It limits processors in two ways: First, only a finite amount of power can be made available to a processor. Second, the power that a processor uses is dissipated as heat. Power has a large effect on a processor's *temperature*. As temperature increases, the

risk of physical failure also increases. A final physical quantity is *area*—the amount of silicon space that a processor requires. Area is usually measured in $mm^2$. More complex processors require more area. Area can be combined with power to measure *power density*, the amount of power dissipated per unit of silicon area. Processors with lower power densities are easier to cool.

Superficially, evaluation is simply a question of the speed of a processor, as well as how much the processor costs to design and operate. The complexity is in the details. While it is not particularly difficult to determine, e.g., how quickly a processor completes a task, determining why the task uses the given amount of time can be much more involved. Processor design is dependent on understanding why a processor behaves as it does. This requires more detailed metrics that can also be less obvious.

### 2.2.2 Rates

Rates can offer more insight into a processor than simply measuring execution time, energy consumption, etc. One of the most obvious rate metrics is *IPS—instructions per second*. Comparing the execution time of two different tasks is not meaningful, as the tasks can have very different behaviors and lengths. Instead, the rates at which the tasks complete instructions can be used to compare their execution speeds. IPS measures speed, but it does not help explain what causes a given speed. *IPC—instructions per (clock) cycle*—is much more helpful in this regard, as it separates the processor architecture from the clock speed. A faster clock can increase IPS but not IPC, and IPC therefore aids in determining whether a performance improvement is due to the architecture or the clock speed. The inverse of IPC, *CPI*, is also used.

*EPI* (*energy per instruction*) provides a similar insight into energy consumption as IPC provides for speed. Simplistically, if there are no memory bottlenecks and there is no leakage power, then the execution of a task on a processor requires a fixed amount of energy. EPI can be used to measure the energy cost of instructions, and energy cost can then be compared across processors. Increasing the clock rate increases IPS. More power is consumed because more instructions are completed every second, but EPI remains constant.

*Throughput* metrics are some of the more involved rate-based metrics. They are used in situations where a number of tasks are competing for resources on a single processor. In such scenarios, the overall throughput of the processor is important, but

so are the individual execution rates of tasks. Throughput metrics are the subject of ongoing research, and are covered in more detail in **section 3.5.2** (p. 33).

### 2.2.3 Summary Metrics

There are two ways in which metrics can be summarized. The first way involves averaging many results together. For example, a designer might execute many different tasks on a processor and calculate the average IPC. Averages produce a single figure of merit, but this necessarily hides variations in the data. It can be helpful to reintroduce some of the lost information by reporting the standard deviation or the range of values, etc. When calculating averages, it is important to use the correct mean. IPC is best summarized with the harmonic mean, since it is effectively a frequency metric that measures the number of events in a unit of time. CPI measures the duration of an event, and should use the arithmetic mean. When the metric is a normalized frequency, like IPC normalized to a baseline, then the geometric mean is the best option.

The second way of summarizing metrics involves combining more than one metric into a single figure of merit. Many authors define summary metrics to establish a value relationship between metrics that cannot otherwise be exchanged. This allows authors to claim that two different processors are equally "good." These metrics often involve energy and delay, and are intended to measure *efficiency*. An extreme example is by Li et al. [110], who define the $EDA^2P$ metric—energy-delay-area-squared product. This metric evaluates processors in the nearly incomprehensible units of seconds-Joules-$mm^{2\times2}$. The metric states, for example, that doubling a processor's execution speed is as valuable as reducing its area by 29%. While such metrics are useful for academic optimization algorithms, it is unlikely that the design cost, practical usefulness, or commercial value of a processor has anything to do with these metrics. One of the themes of this thesis is to argue that multiplying unrelated metrics together leads to metrics that are too abstract to be usable practically. The issue is revisited in more detail in **section 5.6.2** (p. 64).

### 2.2.4 Benchmarking

Metrics are normally evaluated while running *benchmarks*—well-defined, repeatable tasks that represent an activity that a real user would perform with the processor. Benchmarking can take the evaluation process full circle back to physical quantities. A designer might optimize a processor's clock frequency, IPC, EPI, memory bandwidth,

and any number of other metrics. Eventually, though, the designer will need to evaluate whether the processor executes real tasks sufficiently quickly using an acceptable amount of energy while maintaining a reasonable temperature, etc. From a design perspective, questions of how and why a processor behaves as it does are crucial, whereas from a product perspective, overall behavior when running real tasks is far more important than implementation details.

## 2.3 Processor Hardware

The metrics used to evaluate a processor define the important features of the processor. The individual components of a processor must be designed such that their collective behavior optimizes the desired metrics. The following sections present background information and terminology for key features of processor architectures, microarchitectures, and models. Relevant software terminology is also introduced.

### 2.3.1 Architecture

For the purposes of this thesis, *processor* is synonymous with *central processing unit* (*CPU*). On the most basic level, a CPU contains hardware logic for executing a series of instructions. Execution is normally performed by means of an *instruction pipeline*, which processes instructions in stages. A standard pipeline includes stages for fetching instructions from memory, decoding instructions, executing instructions, writing results to memory, and committing completed instructions. The execution stage uses *functional units* (*FU*s) that perform actions dictated by instructions. The most basic type of FU is an *ALU*, an *arithmetic and logic unit*. All computations can be mapped to a series of ALU operations, but most processors contain additional FUs for performing some operations more efficiently. The set of instructions that a processor can execute defines an *instruction set architecture* (*ISA*). Some ISAs support *instruction set extensions*, or *ISE*s, that can be used to add more functionality to a processor.

A CPU with one instruction pipeline can be said to have one processing *core*. Many modern processors are *multicores* or *manycores*, and have more than one CPU core. An entire computer is sometimes referred to as a *system*. A system can contain *GPU*s (*graphics processing units*), *DSPs* (*digital signal processors*), and other peripherals. GPUs and DSPs are beyond the scope of this thesis. A *system-on-*

*chip* (*SoC*) is a physical semiconductor chip that contains a processor and various peripherals. A *multiprocessor system-on-chip* (*MPSoC*) contains a multicore processor and potentially other processing units, like GPUs and DSPs. Communication among the components on an MPSoC often takes place through a *network-on-chip* (*NoC*). NoCs and peripherals are referred to as *uncore* components of MPSoCs. Whether GPUs and DSPs are uncore components is open to debate.

Processors are run at a given frequency, and electricity is provided to a processor at a given voltage. The voltage and frequency of a processor can be changed at runtime. This is called *dynamic voltage and frequency scaling* (*DVFS*). Increasing voltage causes circuits to operate faster, but also increases power consumption. Decreasing voltage reduces power consumption, but must be accompanied by a frequency reduction to avoid timing violations. DVFS has a strong effect on CPU-bound tasks. When the CPU is the bottleneck to completing a task, then DVFS can increase or decrease execution time. The effects of DVFS on memory-bound tasks are much weaker. If execution is dominated by the CPU waiting for data from memory, then DVFS can be used to slow down the CPU without affecting execution time. Some power can be saved in the process. Multicore processors often have more than one voltage and frequency *domain*, and the voltage and frequency of components in each domain are independent of the voltages and frequencies of components in other domains.

### 2.3.2 Microarchitecture

The architecture of a core is dictated by the ISA—the set of instructions that must be supported. There are, however, many possible ways of implementing an ISA. These implementation details make up the microarchitecture of a core.

*Caches* can either be part of a processor's architecture or a core's microarchitecture, depending on whether the cache is shared among cores or used exclusively by one core. Caches store local copies of data from memory for fast access. The first level of on-core cache is usually split into *instruction cache* and *data cache*. The former stores instructions; the latter stores the data that is being worked on. Caches have a number of design parameters, including size, associativity, physical area, access time, etc.

One of the most significant microarchitectural features of a core is whether execution is *in-order* (*IO*) or *out-of-order* (*OoO*). An IO core can only execute instructions in the order in which they appear in the program. An OoO core can reorder instructions during runtime to work around cache misses and other contentions for resources. OoO

cores can also speculate—they can perform some actions before it is known whether the actions are correct and necessary. If the actions turn out to be incorrect, then they must be rolled back. OoO cores are substantially more adept at extracting *instruction level parallelism* (*ILP*) from tasks than IO cores—i.e., they are better at determining which instructions are independent and can be executed in parallel. OoO functionality requires a number of microarchitectural bookkeeping structures, which substantially increases the complexity of the core.

Some of the most significant microarchitectural structures are *registers*, *queues*, and branching logic. The ISA defines a fixed number of registers that a core must have, but an OoO core often contains more physical registers. The core renames registers on the fly to make use of the full physical register file, which allows for greater ILP. An OoO core also makes extensive use of queues. Memory requests wait in a *load queue* and a *store queue* until they can be serviced. Instructions wait in an *issue queue* until they can be dispatched to functional units. A *reorder buffer* tracks all in-flight instructions. Branching logic includes *branch predictors*—tables that predict whether a given control flow branch will be taken or not. Branch predictors are often paired with *branch target buffers*. These are tables that predict the next instruction after a branch.

If all the cores on a multicore processor implement the same ISA, then all cores can execute the exact same tasks. If the cores all have the same microarchitecture, then the processor is *homogeneous*. If the cores have different microarchitectures, then the processor is a *single-ISA*, *heterogeneous* processor. Different cores might be tuned to different types of tasks or optimized to different metrics. A processor with multiple ISAs is also possible. Such processors are discussed in **section 3.2.4.2** (p. 24).

### 2.3.3 Models

Processors are often studied using configurable software models, since implementing real processor hardware is prohibitively expensive and time-consuming. *Simulators* describe processor hardware in software to varying degrees of accuracy. A processor can be simulated by executing the simulator. Hardware is naturally parallel—various hardware components can perform their activities independently and at the same time. Software is much more serial. A simulator must therefore serialize parallel hardware activity. Simulators are inevitably slower than hardware, largely due to the parallelism issue. A second cause of low simulation speed is that simulators execute hardware

descriptions—while hardware simply behaves the way it does, simulators must evaluate a model of hardware behavior.

Simulators can be augmented with *power models*. A power model estimates the energy cost of actions that take place in hardware. For example, reading a cache line or writing a value into the store queue both cost a certain amount of energy. Based on activity counters from a simulator, a power model can evaluate total energy, average power, etc.

Processors are usually designed on the *register transfer level* (*RTL*) using a hardware description language. RTL is *synthesized* down to circuits that are then fabricated. RTL can also be simulated, but RTL is rarely used as an exploratory design tool. An RTL description contains details of all the logic of a processor, which makes modifications difficult and simulation extremely slow.

### 2.3.4 Software Considerations

All processor hardware exists to run software. Given the large variety of software in existence and the numerous ways in which software is analyzed, there exists a broad set of terminology for describing software. This thesis uses *program* and *application* to refer to a complete, self-contained piece of software that a user runs from start to finish. A program is made up of one or more *threads*, or series of instructions. *Multithreaded* programs have more than one thread that can run in parallel, potentially on different cores or even different processors. Some CPU cores support *SMT* (*simultaneous multithreading*), where a single core executes more than one thread in parallel. The threads can be from the same or different programs. A thread can be divided into one or more *phases*. The behavior of the thread is similar for the duration of a phase and distinct from the behavior of other phases. This thesis uses *task* as a generic term for a series of instructions. A task might be an entire, single-threaded program; one thread in a multithreaded program; or one phase of a thread. *Job* refers to a set of programs that are run in parallel, possibly on a compute cluster.

The most fundamental piece of software running on a device is normally the *operating system* (*OS*). The operating system provides services to software, like enabling access to various hardware peripherals. A crucial component of an OS is the *scheduler*. The scheduler controls which thread executes on which core at any given time, and can choose to pause threads or *migrate* them to other cores. The scheduler might work in tandem with a DVFS *governor*. The governor sets DVFS levels. One useful definition

of *task* is the series of instructions that fits into a *scheduling interval*. The scheduling interval is the amount of time that a thread has to execute on a core before the scheduler pauses or migrates it. In some circumstances, the OS must be aware of a program's *deadline*—the time by which the program is required to finish—and the scheduler and governor must respond accordingly.

## 2.4 Summary

This chapter has presented basic processor evaluation and architecture principles. Standard metrics have been defined, and an overview of processor hardware has been given. The remainder of this thesis uses the terminology established here. The next chapter builds on this background information by summarizing the state-of-the-art research in processor design and evaluation.

# Chapter 3:

# Related Work

The problem of heterogeneous processor composition is largely unaddressed in current literature. Heterogeneous processor composition is, however, a natural extension of the existing body of research on processor design. This chapter summarizes the state of the art in research relevant to heterogeneous processors.

## 3.1 Introduction

Heterogeneous processor composition is dependent on five areas of research. The first is the physical architecture of processor hardware—the methods of executing instructions, the methods of storing data, the number of CPU cores, the amount of heterogeneity, etc. The second area is models for analyzing potential hardware before it is implemented. Hardware development is prohibitively expensive and time-consuming, and models must be used to evaluate designs before the investment is made to implement them. The third area is benchmarking. All processor hardware exists to perform some type of computation. The computation can have any number of purposes, from scientific calculations to simply entertaining the user. Regardless of what the purpose is, if the purpose can be expressed as a set of well-defined tasks, then hardware can be designed and benchmarked against these tasks. Benchmarking leads directly to the fourth area: metrics. Metrics are required to quantitatively evaluate how well a processor performs at its intended tasks. The final, perhaps most important, area is the design process—the methods employed to turn the theoretical space of potential processors into one, real processor that has the desired performance on the intended tasks as measured by the metrics. This chapter summarizes the current state of the art in these five areas of research.

## 3.2 Architecture

There are four topics in architecture research that are relevant to heterogeneous processors. The first of these is processor microarchitecture. Modern processors in general, and heterogeneous processors in particular, are subject to strict power and performance requirements. The microarchitecture of CPU cores determines the speed and power of execution. Heterogeneous processors implement different types of cores, where each type has its own microarchitecture. The second topic is dynamic voltage and frequency scaling (DVFS). DVFS is a method of adjusting the power and performance of a core at runtime, and shares many of the goals of heterogeneity. The third topic is homogeneous multicore architectures—processors that implement more than one core, but only one type of core. Heterogeneous multicore processors directly follow from homogeneous multicores. The fourth topic is heterogeneous architectures, which includes research on the different types of heterogeneity and how they can be used.

### 3.2.1 Microarchitecture

Complex, out-of-order (OoO) cores can execute instructions out of program order. Instructions can also be executed speculatively—i.e., before it is known whether the instructions must be executed. These advanced capabilities require a number of microarchitectural structures, which causes the design space of cores to explode in size. Kim and Lipasti [92] describe a number of mechanisms used by cores to replay misspeculated instructions. Mutlu et al. [128] allow execution to continue speculatively in the presence of long stall events. While the speculatively executed instructions must be replayed when the stall event completes, the speculatively executed code prefetches instructions and data, speeding up execution later. Kaynak et al. [89] share prefetched instructions among the cores of a server processor on the premise that different threads of a program are likely to require the same instructions. McFarlin et al. [122] attempt to determine which features of an OoO core are responsible for performance improvements. They argue that most of the performance is due to simply rearranging the order of instructions, and that scheduling around long stall events is only of secondary significance. Czechowski et al. [38] study the sources of performance and efficiency improvements of OoO cores across technology generations. They find that microarchitectural improvements are the major contributor to performance, while the physical scaling of silicon technology is the major contributor to efficiency.

Reddi et al. [148] study the processor features required by a search engine, and argue that a processor with a sufficient number of smaller, more efficient cores can compete with a processor with fewer, faster cores. The authors also argue that a heterogeneous processor is not a good solution for search, as it is not clear when code should run on different types of cores. Rivers et al. [150] explore ways of increasing the bandwidth of caches to support large, OoO cores. Yasin [193] uses hardware event counters to understand where bottlenecks occur in the microarchitecture when executing various tasks.

A more complex microarchitecture has more concurrent activity, and, consequently, higher power consumption. Since modern processors are power-limited, there is considerable research effort into reducing power. Flautner et al. [58] decrease the voltage of individual cache lines to reduce leakage power. Similarly, Homayoun et al. [81] add extra resistance into the clock tree network to reduce leakage current. Ratković et al. [147] study the power, performance, and efficiency trade-offs in adder circuits. Gavin et al. [60] introduce logic to intelligently avoid unnecessary accesses to microarchitectural units in the fetch phase of the instruction pipeline. Liu et al. [113] attempt to determine the amount of energy consumed by individual tasks running on a multicore processor. Arora et al. [6] compare heuristics for determining when a core should go into a low-power state.

The heterogeneous design space used in this thesis is derived from varying the microarchitectural parameters of OoO cores.

### 3.2.2 Dynamic Voltage and Frequency Scaling

Dynamic voltage and frequency scaling (DVFS) is used to adjust the power and performance of a processor during runtime. An overview of DVFS is found in Burd and Brodersen [20]. An early work on DVFS is by Gebotys and Gebotys [62], who set different voltage levels for different components of what is effectively a simple dataflow processor. The aim is to slow down computation that is not on the critical path to lower power consumption.

A number of papers have considered whether “sprinting” or “pacing” is preferable—i.e., whether it is better to quickly finish a task and then power down the processor, or whether it is better to run a task slowly while still meeting deadlines. Choi et al. [31] and Efraim et al. [47] propose DVFS governors to minimize energy. Le Sueur and Heiser [104] describe conditions under which sprinting is preferable

to pacing. Raghavan et al. [146] find that sprinting is generally preferable. Dhiman et al. [43] come to the same conclusion, and note that this is because modern processors effectively implement low-power states. Wang et al. [189] attempt to optimize the DVFS levels of cores when running multithreaded programs, but it is unclear whether the benchmarks used exhibit sufficiently divergent thread behavior to justify the use of different DVFS levels on different cores.

DVFS is often used as a method for controlling the runtime power and temperature of a processor. Su et al. [170] predict the power, energy, and execution time of a running task at different DVFS levels to help determine whether the DVFS level should be changed. Similar approaches could be used to help schedule heterogeneous processors. Pagani et al. [137] determine the amount of power a core can consume based on the amount of power consumed by neighboring cores. If a core's neighbors are not powered, then the core can consume more power without causing a thermal emergency. Cebrián et al. [24] propose several microarchitectural techniques that can be used in conjunction with DVFS to further reduce power and eliminate power spikes.

Providing different voltage levels to a processor is a challenging problem due to the limited number of electrical contacts on a chip. Kim et al. [93] argue for on-chip voltage regulators to increase the number of voltage domains. Mehta and Amrutur [124] suggest using DVFS and also dynamically applying a biasing voltage to the chip bulk silicon to save further power.

The future viability of DVFS is unclear. As transistors continue to scale down in size, the range of usable voltages shrinks. The minimum voltage is expected to stagnate [34], while maximum performance is achieved at ever lower maximum voltages. This trend is observed by Le Sueur and Heiser [103]. Etinski et al. [51] also note the diminishing returns from DVFS, but the reasons for the diminishing returns are not thoroughly explained. From first principles, a heterogeneous processor should be more energy-efficient than a homogeneous processor with DVFS, since heterogeneous cores can be carefully tuned to various performance levels. The intuition is supported by Lukefahr et al. [115]. However, some of the contributions of this work are difficult to apply more broadly, as the authors make optimistic assumptions about both heterogeneity and DVFS. The authors also find a trade-off between performance and energy, which contradicts earlier work from the same group showing that energy is minimized when performance is maximized ("sprinting" [146]).

Due to modeling difficulties and the diminishing benefits of DVFS, DVFS is not included in the design space used by this thesis.

### 3.2.3 Homogeneous Architectures

Homogeneous processors implement duplicates of the same core to increase throughput without increasing the performance of individual cores. Hill and Marty [80] consider the effects of increasing the number of cores on a processor, and come to the fairly obvious conclusion that more research is required on parallelizing programs and increasing the performance of individual cores.

Despite the homogeneity, scheduling tasks to a homogeneous multicore is not necessarily trivial. Bitirgen et al. [13] study the intelligent allocation of CPU, cache, and bandwidth resources to different, concurrently running programs. Ding et al. [45] determine the number of cores that should be allocated to a program and the DVFS level the cores should be set to maximize efficiency. Chen et al. [28] argue that the threads of a multithreaded program running on different cores should consume similar amounts of power. In later work, Chen et al. [29] design logic to determine which sections of parallel programs cause some cores to consume more power than others.

There is a marked difference between homogeneous and heterogeneous processors, as the required design effort increases for every type of core implemented. Eyerman and Eeckhout [53] study whether heterogeneity is required at all. They find that if parallel, scientific programs are used; and if only the throughput of the system must be optimized; and if the processor supports SMT (simultaneous multithreading); then homogeneity is sufficient. Many usage scenarios, including those considered in this thesis, do not match these assumptions.

### 3.2.4 Heterogeneous Architectures

Heterogeneous multicore processors are a natural extension of homogeneous multicores. Motivators for heterogeneity include the better efficiencies of specialized hardware, the limitations of DVFS noted in **section 3.2.2** above, and, perhaps most significantly, the *dark silicon* problem. The dark silicon problem, as described by Esmaeilzadeh et al. [50], is simply that according to current technology trends, transistor size is shrinking faster than transistor power consumption. As a result, processor power density is increasing, and the amount of processor logic that can be powered simultaneously is decreasing. This trend undermines homogeneous processors, since there is little benefit to implementing more identical cores on a processor than can be used concurrently.

Heterogeneous processors work around the problem by having different cores that are not all used at once. There are a number of ways of implementing heterogeneity. The following sections discuss research on single-ISA heterogeneity, multiple-ISA heterogeneity, and reconfigurable processors. The first two types of heterogeneity are heterogeneous in space. Reconfigurable processors are heterogeneous in time, and potentially also in space. While reconfigurable processors are often studied separately from heterogeneity, they face many similar difficulties as fixed heterogeneous processors. The discussions on multiple-ISA heterogeneity and reconfigurability are included for completeness. The technical contribution of this thesis relates primarily to single-ISA heterogeneity, as even this simplest case is poorly understood.

#### 3.2.4.1 Single-ISA Architectures

A single-ISA heterogeneous processor is one where all core types implement the same instruction set architecture (ISA), and the same, unmodified programs can run on all cores. One of the first works on single-ISA heterogeneous processors is by Kumar et al. [96], who use four generations of Alpha cores and run memory-bound workloads on the slower ones. Ren et al. [149] use heterogeneity to minimize energy in server workloads, and argue that the difference between the lowest-power and fastest core types should be maximized. The authors propose a scheduling policy whereby a task begins executing on a low-power core, and is migrated to increasingly faster cores as the task's deadline approaches. Najaf-abadi et al. [133] study a heterogeneous processor composed of identical *nodes*, where each node contains different types of cores. At any given time, only one core is active per node. While the node-based approach may simplify processor organization, it also severely limits the fraction of the computational resource of the processor that can be used simultaneously.

Single-ISA heterogeneous processors that have two types of cores are sometimes called *asymmetric* processors. A common use case for an asymmetric processor is to use a fast core for serial phases of a program, and several slow cores for parallel phases. An example of this is found in Morad et al. [127]. Grochowski et al. [70] argue that EPI (energy per instruction) should be varied during program execution using a combination of asymmetric cores and DVFS. EPI should be higher in a serial program phase and lower in a parallel phase. In a parallel phase, less energy can be expended on each instruction, because many instructions are executed at once. The work is extended by Annavaram et al. [5], who use the processor's power budget to determine what the power and performance difference between slow and fast cores

should be. Van Craeynest and Eeckhout [182] study the trade-off between overall system throughput and the turnaround time for individual programs. They find that a processor needs only to implement two types of cores to achieve any optimal trade-off point between throughput and turnaround time, though the core types vary depending on which trade-off point is required. This work argues strongly for asymmetric processors, and a superficial reading may lead one to believe that no more than two types of cores are ever required. However, the design space used for the work only contains five heterogeneous cores, and the work does not consider power or energy. Broadly applicable conclusions are therefore difficult to draw.

Most commercially available asymmetric processors implement ARM's "big.LITTLE" technology [69]. These include processors from Samsung [152] and Qualcomm [143]. An early asymmetric processor from Nvidia implements cores with identical microarchitectures, but manufactured with different processes [136]. MediaTek has announced plans to add a third core type to a big.LITTLE system [123]. Intel sells an x86-based accelerator under the "Xeon Phi" brand name, which can be used make an asymmetric x86-based system.

Asymmetric processors are *monotonic*, as one core type is always faster and consumes more power than the other. Najaf-abadi and Rotenberg [132] consider running the same task simultaneously on more than one *non-monotonic* core. In this case, different cores maximize performance for different program phases. Cores that lag behind can receive completed instructions from faster cores instead of computing them locally. The approach wastes a substantial amount of power, but can take advantage of fine-grained variations in program behavior to increase performance. Another method of taking advantage of fine-grained variations is proposed by Lukefahr et al. [114], who suggest hybrid "composite cores." The cores have both low-power, in-order fetch logic, and high-power, out-of-order fetch logic. Depending on the type of program phase, the in-order logic can be used to save power without significantly affecting performance. The authors do not, however, satisfactorily address whether the added hardware complexity can be justified, given the area overhead and the associated effects on timing.

Saripalli et al. [155] and Swaminathan et al. [173] consider implementing heterogeneous cores using completely different technologies. They study using low-power cores based on TFET technology along with cores based on standard MOSFET technology.

Scheduling tasks to a heterogeneous processor requires more knowledge of the processor and software than scheduling to a homogeneous processor. Kumar et al. [98] study the scheduling problem for an asymmetric processor. Balakrishnan et al. [9] show that heterogeneity can lead to unpredictable performance if scheduling is poor. Sondag and Rajan [167] analyze compiled binaries of programs for phase behavior and determine whether phases should be run on slower or faster cores. Shifer and Weiss [161] study the scheduling problem for cores that can use either a simple, low-power instruction pipeline, or a complex, high-performance pipeline. Alsafrjalani and Gordon-Ross [4] propose a scheduling algorithm that schedules work based on behavior observed during a learning phase. Challen and Hempstead [25] describe a completely heterogeneous device with two processors, two types of radio, two types of hard disk, etc., and use the flexibility provided by the heterogeneity to adjust which system components use the most power at any given time.

#### 3.2.4.2 Multiple-ISA Architectures

When the cores on a heterogeneous processor are not limited to implementing the same ISA, the number of possible designs for a processor grows considerably. There are several streams of research within multiple-ISA heterogeneity. This area of research is still in the early stages, and it is not clear when the added complexity of supporting different ISAs can be justified.

The simplest multiple-ISA processor is one where cores have partially overlapping ISAs. The instruction sets are identical, except that some cores support additional instructions. Li et al. [111] study how an operating system can make use of a processor with partially overlapping ISAs. When a core encounters an instruction that it does not support, the task is migrated to a core that does support the instruction. Georgakoudis et al. [63] propose dynamically rewriting a program to operate on a given core to avoid the need to migrate the program to another core that supports more instructions. The work leaves unclear what benefits of partially overlapping ISAs can justify the complexity of dynamic binary translation.

Processors with partially overlapping ISAs make use of some cores that support additional instruction set extensions (ISEs). There is a large body of research that blurs the line between ISEs and hardware accelerators. Borkar and Chien [16] argue that increasing power densities require processors with specialized accelerators that are only used when needed. Bingham and Greenstreet [11] suggest finding common dataflow patterns from programs that can then be accelerated with specialized hard-

ware, but include few details. Shao and Brooks [158] search for sections of programs to accelerate in an ISA-independent *IR* (*intermediate representation*). Chien et al. [30] describe the "10x10" project, a project to identify common functionality in programs and to implement that functionality in accelerators. Guha et al. [72, 73] cluster inner loops together based on similarities in the mix of instructions to find code that could potentially be accelerated by the 10x10 project. A limitation of this approach is that there is no guarantee that sections of code with similar instruction mixes can be accelerated by the same accelerator, or that the code can be accelerated at all. Venkatesh et al. [184, 185] describe workflows for extracting frequently used functionality from programs and generating accelerators for that functionality. Goulding-Hotta et al. [68] attempt to develop accelerators for widely-used functionality in the Android operating system. González-Álvarez et al. [67] find commonality between sections of code by expressing the code algebraically. A common limitation of all these works is that the large efficiency improvements announced in headline figures require large numbers of accelerators, but the authors demonstrate their proposals with only a few accelerators. As of yet, there is insufficient evidence to claim that enough code can be accelerated to make these approaches worthwhile.

Some types of operations are known to be common among different programs. For example, Targhetta et al. [174] study hardware for low-power cryptographic operations. Wernsing and Stitt [191] propose software libraries that can run code on accelerators when available. Farmahini-Farahani et al. [57] consider the effects that using CPU cores and accelerators together has on cache behavior.

Some authors consider using two different ISAs together on a processor. Wu et al. [192] suggest a processor with a *VLIW* (*very long instruction word*) core that functions as an accelerator. Binary translation is used at runtime to run code on the VLIW core when possible. The authors argue that the VLIW core can save energy, but the focus of the paper is on code generation rather than on the hardware. DeVuyst et al. [42] show that it is possible to migrate code between a core that implements the ARM instruction set and one that implements the MIPS instruction set, but there does not appear to be any benefit from using these two ISAs together on a processor. Blem et al. [14] compare ARM and x86 cores, and argue that the choice of ISA does not affect power, energy, or performance.

"Invasive computing," a scheduling methodology for manycore processors, is described by Teich [176] and Teich et al. [177]. The methodology can theoretically

make use of many different types of hardware accelerators. Lari et al. [101] discuss controlling power with invasive computing.

There has been considerable research into scheduling work to MPSoCs (multiprocessor systems-on-chip). MPSoCs are systems with many components, like CPUs and various accelerators [118, 140, 144]. These works normally map an audio or video codec, or similar algorithm, to the various components of an MPSoC. It is generally unclear what the significance of this is to real-world problems, since specialized hardware accelerators for codecs are already common. Chandramohan and O'Boyle [26, 27] partition parallel OpenMP programs for an MPSoC, and schedule the partitions to optimize various metrics. Wang et al. [188] determine how much power should be allocated to a CPU and how much should be allocated to a GPU when optimizing the throughput of an OpenCL program.

#### 3.2.4.3 Reconfigurable Architectures

The underlying design goal for reconfigurable processors is the same as the goal for all processors—matching the computational hardware to the computational problem that must be solved. Processors that can be reconfigured at runtime can be more accommodating to the requirements of various types of programs, but this comes at the expense of additional hardware complexity. Kumar et al. [97] study the possibility of sharing caches and floating-point units among cores. While not strictly reconfiguration, the sharing theme is common in later works on reconfigurable processors. DEC and AMD have attempted to share floating-point units between cores in production processors, but AMD has recently moved away from this type of architecture [22]. İpek et al. [85] propose an architecture where smaller, slower cores can be fused together to make larger, faster cores. The intended use case is the same as that assumed by some works on asymmetric processors (**section 3.2.4.1**)—to provide many cores to parallel program phases, and a single, fast core to serial phases. Zhong et al. [195] describe a similar approach of combining small cores into a large core, and develop a compiler to both detect the amount of parallelism available and to control which processor configuration should be used. Gupta et al. [75, 76] propose a processor that contains modular instruction pipeline stages. These are composed into cores during runtime. Research into these types of reconfigurable architectures has stagnated in recent years. This is most likely due to the complexity of reconfiguring hardware combined with the fact that dark silicon makes it possible to simply implement many different cores.

There is ongoing research into dataflow architectures—processors that take advantage of data dependencies in software algorithms and can run independent sections of code in parallel. Nagarajan et al. [130] reconfigure a pool of ALUs (arithmetic and logic units) to match the data flow graph of a program. One of the most widely known dataflow architectures is "TRIPS." In an early work, Sankaralingam et al. [153] use a large TRIPS core that can be reconfigured to function like many smaller cores. Later, Kim et al. [91] use a TRIPS-like architecture where small cores can be reconfigured into a single, large core. Burger et al. [21] describe how TRIPS can be used to execute dataflow graphs. Desikan et al. [41] use TRIPS for speculative execution. Smith et al. [164] present a compiler for TRIPS. Gebhart et al. [61] evaluate an implementation of TRIPS in silicon. Sankaralingam et al. [154] present low-level details of the TRIPS microarchitecture. Putnam et al. [142] describe the "E2" architecture, a successor to TRIPS. Taylor et al. [175] present the "Raw" architecture, which has similar reconfiguration goals as TRIPS, but exposes even more reconfigurability to software.

Rather than configuring cores together in various ways, some works propose specialized, reconfigurable functional units for use within the instruction pipeline. Bauer et al. [10] use such reconfiguration to accelerate a video codec. Cong et al. [35] evaluate a similar architecture for medical imaging applications. Efthymiou and Garside [48] dynamically change the number of stages in an instruction pipeline.

A number of authors have suggested reconfiguring individual structures within a CPU core during runtime. Bahar and Manne [8] adjust the issue width of the instruction pipeline and the number of clock-gated functional units. Zhang et al. [194] similarly adjust the instruction pipeline width to control execution speed and energy consumption. Huang et al. [84] and Sundararajan et al. [171] study cache reconfiguration at runtime. Dubach et al. [46] use a machine learning model to predict optimal sizes for reconfigurable microarchitectural structures. The crucial question with these works is whether the reported benefits of reconfiguration can be realized, given the hardware complexity of reconfiguration. To fully capitalize on the power benefits of resizable structures, the structures must be partitioned, and it must be possible to power-gate individual partitions to eliminate leakage power. This increases the complexity of power delivery circuits and potentially introduces the need for more power domains. Additional logic is also required at partition boundaries and to track which partitions are usable at any given time. The logic used to access the structure must be provisioned such that all partitions can be used. It is unclear if the benefits

of reconfigurability outweigh the power and area overheads. Given the complexity of reconfiguration, reconfigurability must necessarily be coarse-grained, and it may be both cheaper and more efficient to simply implement a few different, static cores.

Some works consider using re-programmable logic along with fixed cores. Dales [39], and Panneerselvam and Swift [138] describe the operating system support required for using reconfigurable logic. Göhringer et al. [65] consider a highly reconfigurable MPSoC that could be implemented on an *FPGA* (*field-programmable gate array*). Kumar et al. [95] automate the generation of MPSoCs for an FPGA.

## 3.3 Models

The general area of architectural modeling can be divided into four research topics. The first is analytical models. These are models that describe the architectural space in terms of high-level, mathematical functions. The second topic is simulators. Simulators model the functionality of hardware in software. Simulators tend to be substantially slower than hardware, because a description of the hardware is evaluated in software. Consequently, the third topic is simulation techniques used to accelerate simulation. The final topic is power models, as power consumption is a top priority for modern processors.

### 3.3.1 Analytical Models

Analytical models can be orders of magnitude faster than simulators, since the hardware is modeled using mathematical functions rather than software descriptions of hardware. Zyuban and Strenski [199] derive equations to relate the energy and delay of CPU core circuits. Lee and Brooks [106] train linear regression models to predict the power and performance of cores. Later, the same authors show that regression models can be used to study the design space of cores [107], and use their models to study regions of the microarchitectural design space where small changes in the size of a structure have large effects on power and performance [105]. Karkhanis and Smith [88] develop analytical models to approximate the Wattch power model. Azizi et al. [7] develop models to study energy-performance trade-offs. This work is one of the few to use hardware models written in a hardware description language (Verilog) in addition to software power models. Blem et al. [15] use first principles to model the theoretical upper-bound of multicore processor performance. Van den

Steen et al. [183] build an analytical model to predict power and performance from the Sniper simulator and McPAT power model.

While significant effort has been invested into these analytical models, their ultimate usefulness is somewhat questionable. The state-of-the-art analytical models are, to a greater or lesser extent, all tuned to software power models like Wattch or McPAT, which are, in turn, based on the CACTI cache model. The most difficult to model component of a CPU core is not cache-like structures, but pipeline logic. Current power models take a naïve approach to modeling pipeline logic, and as a result, there is no evidence that existing analytical models are even capable of modeling real hardware. One might argue that analytical models at least provide a speed-up over simulators and power models, but it is unclear whether this is genuinely beneficial. Analytical models cannot, by definition, model the complex interactions of microarchitectural structures, and they therefore cannot replace detailed simulation. The McPAT power model spends most of its execution time searching the design space of caches. However, the search must only be carried out once per cache configuration, as the results can then be stored in a database. This substantially reduces McPAT's runtime, thereby reducing the evaluation speed benefits of analytical models. Finally, analytical models are presented as alternatives to simulation-based design-space exploration, but the models must first be tuned with simulation results. It is therefore unclear whether analytical models reduce the amount of work required for design space exploration. In light of these considerations, the data used in this thesis is taken directly from simulation without relying on analytical models.

Some authors use analytical models for purposes other than extrapolating simulators and power models. Chung et al. [33] use power, area, and bandwidth models to argue that heterogeneity and accelerators can reduce power, but will not improve performance over a fast CPU. Eyerman et al. [54] study existing x86 processors to determine to what extent various microarchitectural structures affect performance. Nilakantan et al. [135] model the bandwidth requirements of accelerators to determine how much processor area should be allocated to them.

### 3.3.2 Simulators

There exist a number of microarchitectural simulators. These are often called *cycle-accurate simulators*, as they attempt to model real processors accurately down to the clock cycle. gem5 [12] is potentially the most widely-used research simulator

currently available, and is used to generate the data described in **section 4.3** (p. 41). Zoni et al. [197] combine gem5 with several other software tools to model systems-on-chip (SoCs). Gutierrez et al. [78] modify gem5 to be more similar to an ARM development board. Nachiappan et al. [129] build on gem5 to simulate a mobile SoC that also contains accelerator hardware. MARSS [139] is a widely-used x86 simulator that supports heterogeneous multicores. multi2sim [181] simulates both CPUs and GPUs, and can run OpenCL code.

### 3.3.3 Simulation Techniques

Since cycle-accurate simulation is prohibitively slow, considerable effort has been invested into methods for speeding up simulation. Most such methods leverage regularity in program behavior, identify distinct program phases, and selectively execute a subset of a given program. Dhodapkar and Smith [44] compare various phase detection techniques. Marculescu and Iyer [117] use detailed simulation to determine the behavior of a program phase, and then use faster, more approximate simulation to quickly finish simulating the phase. A common simulation approach is "SimPoint" [159, 160]. SimPoint analyzes a program in terms of *basic blocks*—sections of code with one entry and exit point. A series of instructions executed at runtime can be expressed in terms of the constituent basic blocks, and similarities can be identified between different sections of programs. One limitation of SimPoint compared to, for example, the work of Guha et al. [72] in **section 3.2.4.2** above is that SimPoint does not have visibility into basic blocks. SimPoint cannot determine the similarity between the behaviors of basic blocks. Wenisch et al. [190] use statistical sampling to speed up simulation. Jiang et al. [86] use sampling intervals derived from a fractal, but it is unclear why fractal-based sampling is superior to statistical sampling.

Hoste et al. [83] depart from statistical sampling, and instead use program similarity to predict performance. A program's performance is estimated based on its similarity to programs with known performances. Eklov et al. [49] build statistical models of cache miss rates to predict program performance. Leupers et al. [109] present an overview of a number of techniques for speeding up simulation.

These advanced simulation techniques can substantially reduce the amount of time spent on cycle-accurate simulation. Depending on how the experiments are designed, the techniques can also have a positive or negative effect on the statistical significance of results. While the amount of time required for the simulations used by this thesis is

large, it is not prohibitive. In light of the increased complexity and risk of statistical simulation, the experiment methodology (**section 4.3** (p. 41)) does not use statistical sampling.

### 3.3.4 Power Models

Simulators predict the performance of a given program on a given hardware configuration, but power models are required to estimate power and energy consumption. One of the most widely used power models is the CACTI cache model [178]. CACTI forms the basis of the Wattch power model [19], as well as the more recent McPAT power model [110] used in **chapter 4**. Spiliopoulos et al. [168] use data from McPAT to model DVFS with the gem5 simulator. Keckler et al. [90] study the energy cost of GPU calculations and memory accesses. Shye et al. [162] use smartphone usage data to predict where the device spends its power and where power consumption can be reduced.

## 3.4 Benchmarking

There are two aspects to benchmarking: benchmark suites and benchmarking techniques. One of the most widely used benchmark suite in the microarchitectural community is the SPEC CPU suite [169]. While SPEC is not necessarily suitable for academic research [79], it continues to be a crucial component of evaluation methodologies. Newer benchmark suites, like EEMBC [141] and BBench [77] have been created to evaluate mobile processors. The data set in **section 4.3** (p. 41) uses SPEC and EEMBC benchmarks.

A number of works consider the best way to benchmark processors. Hoste and Eeckhout [82] analyze static benchmark features to identify unique and redundant benchmarks. Meanwhile, Najaf-abadi and Rotenberg [131] argue that excluding benchmarks from a suite based on benchmark similarity can harm the processor being designed. Breughe and Eeckhout [17] study various ways of selecting which dataset to use with a benchmark. Gupta et al. [74] suggest a method for evaluating the CPU usage of individual programs running together on a multicore processor. Curtsinger and Berger [37] randomize the layouts of benchmarks in memory to increase confidence that benchmarking results are statistically significant.

## 3.5 Metrics

There is a large body of research considering the best metrics for evaluating various aspects of processors. In some cases, the question of the correct metric to use is a matter of ongoing debate. New metrics are defined when quantities like performance, energy, and other measurable features do not provide sufficient insight. More complex metrics attempt to quantify more complex features of processors. There is always a level of subjectivity involved in defining metrics, and consequently, there is an almost inevitable debate regarding whether a given metric truly measures the concept it is intended to measure, and whether the concept is truly worth considering in the design of a processor. The following sections summarize work on energy-aware metrics, throughput metrics, uniformity metrics, and statistical techniques.

### 3.5.1 Energy-Aware Metrics

Some of the most common metrics measure efficiency using an energy-delay product. The intuition behind these metrics is fairly obvious: Fast processors—ones with low delay—are desirable, as are processors that consume little energy. Multiplying energy and delay together and optimizing the product allows the two quantities to be traded off and balanced. Gonzalez and Horowitz [66] are possibly the first to suggest the energy-delay product metric (ED). Brooks et al. [18] suggest variations on ED. These include the power-delay product, which is mathematically equivalent to energy, and $ED^2$, a metric that gives a double weight to delay. Martin et al. argue forcefully for the $ED^2$ metric [119, 121]. $ED^2$ emerged during the design of an asynchronous processor [120]. As also noted by Brooks et al. [18], $ED^2$ is voltage-invariant—it is independent of supply voltage for CPU-bound tasks on normal processors, and for asynchronous circuits. Some authors use $IPS^3/W$ (instructions-per-second-cubed per Watt) [46, 71, 107, 134]. $IPS^3/W$ is inversely proportional to $ED^2$. Alioto et al. [3] use energy-delay metrics with a number of different weights for the energy and delay terms.

Since energy-delay metrics were first proposed in the 1990s, they have become some of the most widely used metrics in the literature [7, 17, 23, 25, 27, 30, 45, 53, 58, 59, 84, 96, 103, 110, 147, 155, 161, 170, 173, 174, 178, 184, 185, 194, 198, 199]. Energy-delay metrics are not, however, without complications. Sazeides et al. [157] show that optimizing ED or $ED^2$ on a program phase granularity can lead to sub-

optimal global results. Energy-delay metrics also implicitly assume that there is a trade-off between energy and delay, but sometimes the fastest option also minimizes energy [31, 38, 64, 146]. **Section 5.6.2** (p. 64) argues that energy-delay metrics are not an appropriate choice for heterogeneous processors.

Another common energy metric is EPI—energy per instruction [5, 88]. EPI can often be easier to interpret than energy-delay metrics, as it simply evaluates the energy cost of executing the average instruction. EPI can be combined with performance metrics, like IPS (instructions per second), to understand both execution speed and energy cost. EPI and IPS multiplied together produce runtime power [70].

### 3.5.2 Throughput Metrics

Most modern processors can execute multiple tasks in parallel by using SMT, by using multiple cores, or by using both. This has led to the definition of throughput metrics to evaluate performance. Snavely and Tullsen [165] define the *weighted speedup* metric to measure the throughput of single-core processors with SMT. In a later paper, Snavely et al. [166] evaluate the throughput of various thread scheduling policies for SMT processors. Optimizing for throughput maximizes the total work completed by a processor, but can have negative effects on individual tasks. Luo et al. [116] evaluate both the throughput and fairness of SMT thread scheduling. Eyerman and Eeckhout [52] rename the weighted speedup metric to *STP* (*system throughput*), and introduce *ANTT* (*average normalized turnaround time*). STP and ANTT are intended for multicore processors. ANTT is a fairness metric to counterbalance STP, since increasing the overall throughput of a processor also increases the average execution time seen by individual programs. Eyerman et al. [55] define several throughput metrics that are applicable to various types of experiments. Michaud [125] discusses some of the pitfalls of throughput metrics, and shows that some definitions of throughput lead to inconsistencies. Throughput metrics are appropriate for servers and other fully subscribed processors, but less applicable to mobile devices (see **section 5.6.1** (p. 63)).

### 3.5.3 Uniformity Metrics

Processor design involves trading off various quantities. A number of metrics have been defined by researchers in the field of multiobjective optimization to evaluate points sampled from optimal trade-off curves. The relevance of these metrics to heterogeneous processors is discussed in detail in **section 6.6.5** (p. 88). Sayın [156]

defines ε-coverage error, δ-uniformity, and cardinality metrics. Laumanns et al. [102] define ε-dominance. Farhang-Mehr and Azarm [56] use entropy to measure diversity. Deb et al. [40] define a crowding distance metric and the Δ-nonuniformity metric. Ackerman and Ben-David [1] study "clusterability." While not directly relevant to multiobjective optimization, "clusterability" evaluates how well points have been clustered, and is related to uniformity and diversity metrics.

### 3.5.4 Metrics from Statistics

A number of statistical techniques are relevant to processor design in general, and the composition of heterogeneous processors in particular. Conover [36] presents nonparametric statistics, including Spearman's ρ and the KS test. These are used in **chapter 6**. Rutherford [151] gives a thorough overview of analysis of variance (ANOVA) techniques. Vinh et al. [186] summarize a number of variations of the mutual information metric. Vuduc et al. [187] define an early stopping criterion that can be used to evaluate the quality of a random search of a design space. **Chapter 4** makes use of ANOVA, mutual information, and Vuduc's criterion.

## 3.6 Design Process

There are two aspects to heterogeneous processor design: design space exploration (DSE) and core selection. DSE methods are used to characterize the space of possible CPU cores and to find cores that could be used on a heterogeneous processor. Core selection is used to determine which cores should be used on the processor. The difference between DSE and selection is largely unrecognized in the literature. DSE has received significant research interest, while selection is often ignored and generally not even identified as a problem. This may be because of an implicit assumption that in an academic context, DSE methods are a sufficiently significant contribution, and that core selection is not possible without access to restricted, industrial information. The premise of the current thesis on heterogeneous processor composition is that this assumption is untrue, and that more progress can be made on the core selection problem using only freely available data.

### 3.6.1 Design Space Exploration

In an early work, Zyuban and Kogge [198] study performance-energy trade-offs for out-of-order processors. The authors suggest that the trade-offs allow one to select the fastest core given a power budget. Since the work does not attempt to define a power budget but assumes that one is provided, the work, by itself, cannot be used to select a core. Similar assumptions are present in many later works—it is often assumed that there exist criteria external to the DSE process that are theoretically knowable but unknown to the researchers, and that these criteria can be used to easily select cores from the characterized space. For research on DSE methods to be practically useful in the absence of core selection methods, this assumption must be true. However, few works, if any, address the assumption.

Kang and Kumar [87] use machine learning to reduce the number of simulations required for design space exploration. Turakhia et al. [180] search the design space of heterogeneous cores using multithreaded benchmarks. Sunwoo et al. [172] present a method of searching the design space of a smartphone processor using graphical benchmarks. Lee et al. [108] propose a DSE method that can model the performance of many different cores based on results gathered during only one simulation run. Choudhary et al. [32] describe the FabScalar infrastructure of out-of-order core components. Unlike most simulators, FabScalar is based on RTL models that can be synthesized to hardware. As a result, FabScalar can enable more accurate DSE. Liu et al. [112] describe a method of DSE that relies on repeatedly synthesizing a given set of hardware components.

Some authors have explored the design space of entire systems-on-chip (SoCs) rather than just the space of possible cores. Cassidy et al. [23] design an analytical model to predict the fraction of a processor's area that should be used for cores, the fraction for caches, and the fraction for interconnects. Givargis et al. [64] propose a DSE method for SoCs that quickly prunes configurations that are not power-performance optimal. Mishra et al. [126] explore the space of heterogeneous networks-on-chip (NoCs) for use with SoCs.

Design space exploration is a special case of multiobjective optimization. A processor designer prefers cores that are fast, consume little power, require a small amount of silicon area, etc., but many such requirements are mutually exclusive. Multiobjective optimization algorithms search for the *Pareto optimal* trade-offs between competing objectives—i.e., the solutions where no objective can be further improved without

another objective deteriorating. Kursawe [100] motivates multiobjective optimization by arguing that in many cases, there is no function that can combine multiple objectives into a single objective, and all possible trade-offs between the objectives are therefore potentially useful. This observation is particularly relevant to DSE for CPU cores: There is, for example, no objectively correct equation for trading power for performance. Metrics like $\text{IPS}^3/\text{W}$ assume that performance is three times more valuable than power. This assumption is arbitrary, and some cores may require other trade-offs. The designer should be aware of all possible trade-offs between power and performance, not just the one that optimizes $\text{IPS}^3/\text{W}$. I.e., the designer should have access to the power-performance Pareto-optimal set of cores.

A number of algorithms have been proposed to efficiently find Pareto-optimal sets. Zitzler and Thiele [196] present an overview of multiobjective evolutionary algorithms (MOEAs) and define the SPEA optimization algorithm. Knowles and Corne [94] define the PAES algorithm, and Deb et al. [40] define the NSGA-II algorithm. Adra and Fleming [2] present modifications to NSGA-II to increase the diversity of the solution set.

This thesis does not attempt to add to the work on DSE. **Chapter 7** only assumes that a processor designer has access to a method of design space exploration.

### 3.6.2 Core Selection

A number of authors select heterogeneous cores to optimize a single metric. Kumar et al. [99] select cores to optimize performance using an exhaustive evaluation of all possible sets containing a given number of cores. Lee and Brooks [107] select cores to optimize $\text{IPS}^3/\text{W}$ and then reduce the number of selected cores to the desired number using clustering. Navada et al. [134] use a genetic algorithm to search a design space and select a given number of cores to optimize performance. All three works include the option of selecting cores to meet a given power budget. Singh et al. [163] also perform selection, but do so for an MPSoC that is specialized for audio and video applications.

The work of Guevara et al. [71] is unique in the literature, as the authors select cores to implement different power-performance points instead of optimizing a single metric. Similar cores are clustered together, and the most consistent core is selected from each cluster. Various consistency metrics are considered. Selection methods that optimize for one metric result in non-monotonic heterogeneity, whereas the method

of Guevara et al. [71] is possibly the only published work to select monotonically heterogeneous cores from a large candidate set. As a result, the selected cores can run tasks at various power-performance points. **Chapter 8** evaluates this "Clustering" core selection strategy.

## 3.7 Summary

Heterogeneous processors contain CPU cores with different microarchitectures and potentially even different architectures. During the design process, processors must be modeled, benchmarked, and evaluated. This chapter has summarized current research in these and related areas.

# Chapter 4:

# Infrastructure

This chapter describes the software tools that generate the example dataset used to demonstrate the techniques in later chapters. The experimental methodology underlying the dataset is part of the infrastructure.

## 4.1 Introduction

Methods for composing and evaluating heterogeneous processors should be decoupled from the specific implementation details of the processor being designed. Such methods cannot be too closely related to any particular processor design space or application space, lest they lose their applicability to other design and application spaces. This chapter describes one particular software infrastructure that is used to generate an example dataset. The selection and evaluation techniques in later chapters are demonstrated with this dataset. The later chapters are not, however, dependent on this specific infrastructure. A different, equally valid dataset could be generated by changing any number of parameters in the experimental methodology, or exchanging any of the software tools for other tools.

There are two components to the infrastructure: the software tools used to generate the dataset and the experimental methodology by which the dataset is generated. The following sections discuss these in turn.

## 4.2 Software Tools

The infrastructure is based on four software tools: the gem5 cycle-accurate simulator, the McPAT power model, the wrapper scripts that run simulations, and the R statistical package. Each tool is described below.

### 4.2.1 gem5 Simulator

gem5 is a highly configurable, cycle-accurate microarchitectural simulator [12]. The example dataset uses development version 9351:4229aedfdd09 of gem5. gem5 is built for the ARM ISA and run in system call emulation mode (SE). The "arm_detailed" out-of-order CPU core model forms the basis of the simulations. gem5's standard simulation scripts are modified such that microarchitectural parameters can be overridden by a user-specified file.

### 4.2.2 McPAT Power Model

The McPAT power model [110] is largely an interface to CACTI cache models [178]. The example dataset uses version 0.8 of McPAT. McPAT is configured through an XML interface [179]. The configuration is adjusted to match gem5's "arm_detailed" model as closely as possible. McPAT is run with the "optimize for clock" option for a 28nm, low-power process technology.

### 4.2.3 Simulation Scripts

Each run of gem5 and McPAT is launched from a perl script. gem5 is run on a compute cluster as an array job. The cluster scheduler invokes the simulation script once for each index in the job array. The simulation script generates a configuration file from the array index and launches gem5. The simulation script is configured such that each benchmark is run with the same gem5 configurations, but the configurations are randomly sampled from a large space of microarchitectures. One gem5 run can take anywhere from a few minutes to a few hours, depending on the benchmark.

McPAT is run separately from gem5 using the simulation data generated by gem5. McPAT has a fixed runtime of a few seconds, which is not dependent on the length of gem5 simulations. For a large experiment, gem5 requires compute-years of time, but McPAT only requires compute-days. Consequently, the McPAT script is run on a single machine, and McPAT is invoked for each simulation result in series.

Once gem5 and McPAT have run, a final script parses the logs and output files, and builds a table of results. The table is then loaded into R for analysis.

| SPECint 2006 | | |
|---|---|---|
| astar | h264ref | omnetpp |
| bzip2 | hmmer | perlbench |
| gcc | libquantum | sjeng |
| gobmk | mcf | xalancbmk |

**Table 4.1:** ***SPECint 2006*** *benchmarks [169]*

| EEMBC | | |
|---|---|---|
| aes | mp3player | qos |
| cjpegv2 | mpeg2decode | rgbcmykv2 |
| des | mpeg2encode | rgbhpgv2 |
| djpegv2 | mpeg4decode | rgbyiqv2 |
| huffde | mpeg4encode | routelookup |
| ip_pktcheck | nat | rsa |
| ip_reassembly | ospfv2 | tcp |

**Table 4.2:** ***EEMBC*** *benchmarks [141]*

### 4.2.4 R Analysis

All analysis is carried out using the R statistical package [145]. Unless otherwise noted, algorithms and mathematical functions are implemented in R. All standard statistical techniques use the implementation in R where available. Figures showing quantitative data are also generated using R.

## 4.3 Experiment Methodology

The example dataset contains data on a set of benchmarks that have been simulated on a set of cores. The benchmarks and the design space of cores are described first, followed by a discussion on the dataset.

### 4.3.1 Benchmarks

The example dataset uses two benchmark suites: the SPEC 2006 integer benchmarks, listed in **table 4.1**, and the EEMBC benchmarks, listed in **table 4.2**. The EEMBC suite combines the EEMBC DENBench (digital entertainment) and the EEMBC Network-

ing 2.0 benchmarks. The EEMBC suite contains smaller benchmarks that represent common tasks on mobile devices. The SPEC suite contains more demanding scientific benchmarks. The two suites are always analyzed independently of each other.

All benchmarks are compiled for the ARM ISA using gcc with the `-O3` option. Benchmarks are linked statically as required by gem5's system call emulation mode. SPEC benchmarks use the "training" datasets. EEMBC benchmarks are simulated in detail for one billion instructions or until completion, whichever comes first. EEMBC benchmarks have small memory footprints, with median instruction and data cache miss rates of <0.1% and 1% respectively. To reduce simulation time, they are simulated without an L2 cache. For the much larger SPEC benchmarks, the first two billion instructions are fast-forwarded. There is a warm-up period of ten million CPU cycles, followed by one billion cycles of detailed simulation. Data is only collected for the one billion cycles of detailed simulation. To reduce runtime, the SPEC benchmarks use a check point saved at two billion instructions. SPEC benchmarks have median instruction and data cache miss rates of 2% and 5% respectively, and are run with a 2MB, 8-way L2 cache. The total gem5 runtime for the example dataset is seven compute-years for the EEMBC suite and nine compute-years for the SPEC suite.

There is endless debate regarding which benchmark suites are the "correct" ones to use in a given setting. Ultimately, a good benchmark suite is one that is *representative* of the full range of applications that will be run on the processor that is being designed. *Representative* can mean many different things, but will likely include considerations like instruction mix, memory behavior, the relative frequency of different types of computation, etc. Workload characterization is beyond the scope of this thesis, and it is simply assumed that the SPECint 2006 suite and the EEMBC suite are each representative of some application space. Two different suites are used to demonstrate that the techniques in later chapters react to differences in application spaces.

### 4.3.2 Design Space

The example dataset contains data for a design space of 3000 different CPU cores. The data includes the execution time and runtime power (combined dynamic and leakage) for each of the 33 benchmarks in **section 4.3.1**. The design space is a randomly sampled subset of a microarchitectural space containing over 30 billion permutations of cores. It would be possible to use a more intelligent sampling method than random. However, random sampling has two advantages: First, it avoids any (potentially

| Parameter | | Values |
|---|---|---|
| Data Cache | Size | 16, 32, 64 kB |
| | Associativity | 1, 2, 4 |
| Instruction Cache | Size | 4, 8, 16, 32, 64 kB |
| | Associativity | 1, 2, 4 |
| Registers | Integer | 50, 64, 96, 128, 256 |
| | Floating Point | 96, 128, 256 |
| Queue Entries | Issue | 16, 32, 64 |
| | Load | 8, 32, 64 |
| | Store | 8, 16, 32, 64 |
| | Reorder Buffer | 16, 32, 40, 64, 128 |
| Branch Predictor | Global Counter Bits | 1, 2, 3 |
| | Global Entries | $2^{10}$, $2^{11}$, $2^{12}$, $2^{13}$, $2^{14}$ |
| | Local Counter Bits | 1, 2, 3 |
| | Local History Bits | 10, 11, 12 |
| | Local History Entries | $2^{9}$, $2^{10}$, $2^{11}$ |
| | Choice Counter Bits | 1, 2, 3 |
| | Choice Entries | $2^{10}$, $2^{11}$, $2^{12}$, $2^{13}$, $2^{14}$ |
| Branch Target Buffer | Entries | $2^{10}$, $2^{11}$, $2^{12}$, $2^{13}$, $2^{13}$ |
| | Tag Bits | 16, 18, 20 |

**Table 4.3:** *Summary of microarchitectural parameter values used in the example design space*

unknown) biases in a sampling algorithm that could affect later results. Second, by including simulation data on even some very poor cores, random sampling provides a more complete picture of the microarchitectural space. This makes drawing statistically significant conclusions about the design space simpler, and aids later analysis on selection algorithms (**chapter 8**).

Of the 3000 cores in the design space, 266 are power-performance Pareto-optimal for at least one SPEC benchmark, and 363 are Pareto-optimal for at least one EEMBC benchmark. The set of 266 cores and the set of 363 cores are *candidate sets*. **Chapter 7** and **chapter 8** consider the problem of selecting cores from these candidate sets.

The microarchitectural parameters that are varied in the design space are listed in **table 4.3**. The complete list of all 3000 simulated cores is in **appendix A**. Cores are

| Sampling Confidence (SPECint 2006 — 2% Tolerance) | | | | | |
|---|---|---|---|---|---|
| **Benchmark** | **Confidence** | | **Benchmark** | **Confidence** | |
| | Speed | Power | | Speed | Power |
| astar | 1.00 | 1.00 | libquantum | 1.00 | 1.00 |
| bzip2 | 1.00 | 1.00 | mcf | 1.00 | 1.00 |
| gcc | 0.86 | 1.00 | omnetpp | 0.95 | 1.00 |
| gobmk | 0.95 | 1.00 | perlbench | 0.86 | 1.00 |
| h264ref | 0.99 | 1.00 | sjeng | 0.63 | 1.00 |
| hmmer | 1.00 | 1.00 | xalancbmk | 1.00 | 1.00 |

**Table 4.4:** *Confidence that the best power and performance discovered by random sampling is within 2% of the best possible power and performance available in the entire microarchitectural space of 30+ billion permutations (**SPECint 2006** benchmarks).*

numbered from #1 to #3000, but as the configurations are randomly sampled, a core's number has no relationship to its microarchitectural parameters. The microarchitectural space only contains parameters that can be reliably adjusted in both gem5 and McPAT. McPAT's modeling of some parameters, like issue width, is extremely poor. McPAT relies on CACTI cache models, but the issue pipeline stage and similar structures are dominated by logic rather than storage. This makes comparisons among cores with different issue widths highly suspect. Issue width and other such parameters are consequently excluded from the space, along with parameters that cannot be changed in both gem5 and McPAT. McPAT is also unable to model DVFS, so voltage-frequency levels are excluded from the design space. All cores are clocked at 1GHz.

### 4.3.3 Discussion

Two questions must be addressed regarding the experimental methodology: The first is whether the design space of 3000 randomly sampled cores is sufficiently large, given that out of the microarchitectural space of billions of permutations, fewer than one core for every million is included in the design space. The second question is the significance of the microarchitectural parameters. The design space includes all parameters that can be varied reliably in both gem5 and McPAT, but an adjustable parameter is not necessarily a significant one.

| Sampling Confidence (EEMBC — 2% Tolerance) | | | | | |
|---|---|---|---|---|---|
| **Benchmark** | **Confidence** | | **Benchmark** | **Confidence** | |
| | Speed | Power | | Speed | Power |
| aes | 0.86 | 1.00 | mpeg4encode | 1.00 | 1.00 |
| cjpegv2 | 1.00 | 1.00 | nat | 1.00 | 1.00 |
| des | 1.00 | 1.00 | ospfv2 | 1.00 | 1.00 |
| djpegv2 | 1.00 | 1.00 | qos | 1.00 | 1.00 |
| huffde | 1.00 | 1.00 | rgbcmykv2 | 1.00 | 1.00 |
| ip_pktcheck | 1.00 | 1.00 | rgbhpgv2 | 1.00 | 1.00 |
| ip_reassembly | 1.00 | 1.00 | rgbyiqv2 | 1.00 | 1.00 |
| mp3player | 1.00 | 1.00 | routelookup | 1.00 | 1.00 |
| mpeg2decode | 1.00 | 1.00 | rsa | 1.00 | 1.00 |
| mpeg2encode | 1.00 | 1.00 | tcp | 1.00 | 1.00 |
| mpeg4decode | 0.99 | 1.00 | - | - | - |

**Table 4.5:** *Confidence that the best power and performance discovered by random sampling is within 2% of the best possible power and performance available in the entire microarchitectural space of 30+ billion permutations (**EEMBC** benchmarks).*

**Random sampling.** If the sample of 3000 randomly selected cores is sufficiently large to be representative of the microarchitectural space of 30+ billion permutations, then sampling and simulating more cores from the microarchitectural space will not improve the quality of the sample. Conversely, if sampling more cores increases the quality of the sample, then the sample of 3000 is not sufficiently large. The problem of determining how many random samples are enough is addressed by Vuduc's early stopping criterion [187]. The criterion estimates the likelihood that further random sampling will lead to a better result based on the distribution of existing samples. **Table 4.4** and **table 4.5** list confidence values from Vuduc's criterion for the 2% margin of performance and power for each benchmark. I.e., this is the confidence that the best (fastest) performance and best (lowest) power of the 3000 sampled cores are within 2% of the best possible in the entire microarchitectural space. For most benchmarks, the confidence values are 1.0 or very close. It is very unlikely that a better result could be found in the microarchitectural space. Exceptions are the performance confidence values for *aes*, *gcc*, *perlbench*, and *sjeng*. *sjeng* has a particularly low confidence, at

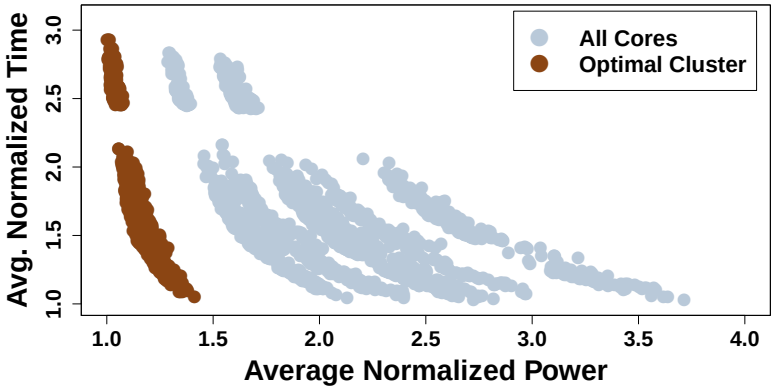


**Figure 4.1:** *Most power-performance Pareto-optimal cores are clustered into one region of the microarchitectural space. Data is for the **EEMBC** suite.*

only 0.63—there is 37% chance that the best performance in the design space is not within 2% of the best performance possible in the microarchitectural space. However, confidence is low not because the sample of 3000 is too small, but because a 2% margin is a high standard. For a 5% margin—the likelihood that best performance is within 5% of the best possible—confidence values for the four benchmarks increase to 1.00, 0.95, 1.0, 1.0, respectively. It can be concluded that the random sample of 3000 cores provides statistically significant coverage of the microarchitectural space for all benchmarks, and for most benchmarks, confidence is well above a 5% margin.

**Parameter analysis.** All the analysis in this thesis is based on the power and performance of cores relative to other cores. None of the analysis is dependent on how the cores at various power-performance points are implemented—the analysis is isolated from implementation details, and treats the implementation as a black box. This ensures that all techniques are portable to other design spaces and are not dependent on the correctness of any particular set of tools. If gem5 and McPAT are found to have fundamental modeling flaws, for example, then the flaws can be fixed or the tools can be replaced without affecting the analysis techniques. It is helpful, however, to understand the features of the example dataset, as this provides further insight into the selection algorithms in **chapter 7** and **chapter 8**.

**Figure 4.1** plots average power and performance for all 3000 cores for the EEMBC suite. The results are largely the same for the SPEC suite. For parameter analysis purposes, the significant feature is the power-performance Pareto-optimal cluster at the left of the plot. Mutual information [186] can be used to show that this cluster is completely explained by three microarchitectural parameters: data cache size, data cache associativity, and integer register file size. The necessary and sufficient features

that cause a core to be in the optimal cluster are a 16kB or 32kB data cache, a 1-way or 2-way data cache, and an integer register file with 128 or fewer entries. If the data cache is larger, if it is 4-way associative, or if the integer register file has 256 entries, then the core becomes substantially more complex and requires much more power. Some cores that are not in the optimal cluster are power-performance Pareto-optimal for a few benchmarks, but the number of benchmarks that can effectively use the largest microarchitectural structures is small.

ANOVA (analysis of variance) [151] can be used to determine which microarchitectural parameters control the power-performance trade-off within the optimal cluster. The integer register file has the largest effect, as it is instrumental in exposing instruction-level parallelism (ILP). Other parameters have less consistent effects. For example, for the *aes* benchmark, when there are 64 integer registers, then instruction cache size has the second-greatest effect, but with 96 integer registers, the number of local history table entries has the second-greatest effect. Queues tend to also have a large effect, while most of the branch predictor parameters have statistically insignificant effects.

## 4.4 Summary

The crucial component of the infrastructure is the example dataset that is used as a basis for demonstrations throughout the remainder of the thesis. This chapter has described the software tools, benchmarks, and microarchitectural space used to generate the dataset.

# Chapter 5:

# Technical Motivation

The heterogeneous CPU core selection and evaluation problems form a barrier between design space exploration and the production of a heterogeneous processor. The significance and difficulty of the two problems are widely overlooked by current literature—it is often assumed that only trivial effort is required to transform the results of a design space exploration into a heterogeneous processor. This chapter establishes a theoretical framework for reasoning about the selection and evaluation problems by defining requirements for power-limited, heterogeneous processors; describing a runtime model for analyzing processors; and demonstrating the deficiencies of current techniques in meeting the requirements.

## 5.1 Introduction

The closely related problems of selecting and evaluating sets of heterogeneous cores introduce a number of considerations that must be addressed. Some of these considerations appear trivial, but substantially complicate the selection and evaluation problems. Other considerations appear intractable at first, but can be surmounted using correctly chosen simplifying assumptions. This chapter lays the groundwork for solving the selection and evaluation problems for mobile devices by establishing the significance of the problems and detailing the shortcomings of existing solutions.

The motivation for the selection and evaluation problems is presented in six parts. First, runtime *flexibility* is established as the ultimate goal for a heterogeneous processor. A heterogeneous processor has the potential to achieve flexibility through microarchitectural *diversity*. Second, the problems associated with diversity are detailed. These are related to both selecting a diverse set of heterogeneous cores, as well as evaluating the diversity of a set of cores. The major complicating factor

to diversity in the processor hardware is correctly handling the diversity in software behavior. Third, a theoretical runtime model is described. This model forms the basis of a framework that enables the analysis of even very large heterogeneous processors with many cores and many tasks running in parallel. Fourth, it is argued that while diversity is intuitively obvious, it cannot be readily evaluated mathematically. Fifth, metrics currently available for evaluating processors are discussed. It is shown that these metrics do not adequately evaluate diversity. A substantial component of the discussion is devoted to energy-delay summary metrics, as these are widely used and often misused by the architectural community. Sixth, existing core selection techniques are summarized. These techniques are shown to not provide a satisfactory level of flexibility.

## 5.2 Flexibility: A First-Order Requirement

Mobile processors must perform well on many different types of programs while being constrained to strict power budgets. Since mobile devices depend on a battery and generally use passive cooling, the amount of power that a program can be allowed to consume is limited, and varies depending on battery charge, ambient conditions, and other running programs. Unlike a server, a modern eight-core mobile device is likely to be under-subscribed. Despite these considerations, users of mobile devices have high expectations for performance. If a processor is *flexible*, then the operating system scheduler can run a given program with the best possible performance, taking into account the program's priority, thermal considerations, the charge left in the battery, etc. Traditionally, the flexibility to choose between high-performance or low-power operation has come from DVFS (see section **section 3.2.2** (p. 19)). However, since each core in a heterogeneous processor is tuned to a particular power-performance point, heterogeneity has the potential for greater flexibility and better energy efficiency than DVFS [115]. Furthermore, DVFS becomes increasingly difficult to implement as silicon technology scales [103]. Heterogeneity can be combined with DVFS, but due to power model limitations (see **section 4.2.2** (p. 40)), this discussion only considers heterogeneous cores without DVFS. The discussion remains valid if it is applied to power-performance points derived from a combination of heterogeneity and DVFS rather than just heterogeneity.

In the case of a homogeneous processor, the designer considers a number of different CPU core designs with different power-performance trade-offs, and chooses

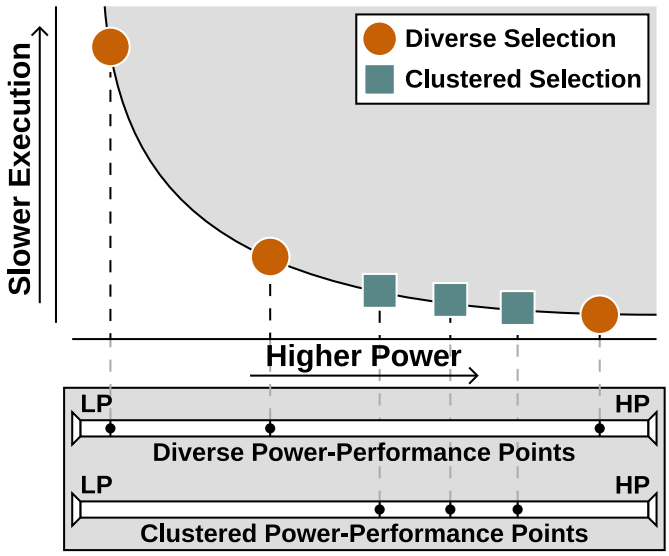


**Figure 5.1:** *Cores selected for flexibility (circles) implement diverse power-performance points, ranging from low power (**LP**) to high performance (**HP**). If cores have similar power-performance points (squares), then there is little flexibility available at runtime.*

to implement one of them. In this case, the scheduler has access to only limited flexibility through DVFS. In contrast, in the case of a heterogeneous processor, the designer selects more than one power-performance point (more than one type of core) to implement. The designer leaves it to the scheduler to make the final choice about which CPU core is best, because the scheduler has more information about what is required of the CPU at a given time. This is the underlying goal of heterogeneity—to give the scheduler more choice (flexibility) at runtime using a diverse set of cores.

A crucial observation is that not all selections of heterogeneous cores are diverse. As an example, **figure 5.1** contains six cores that are all power-performance Pareto-optimal—each core implements a unique power-performance point, and no core has both better power and better performance than any of the others. Runtime flexibility is maximized if the processor designer implements all six cores, but this may not be possible. If the three "diverse" cores are implemented, then the selection is still flexible, as there are diverse power-performance points. If, instead, the "clustered" cores are implemented, then the selection is not flexible. If only the "clustered" cores are implemented, then the scheduler can only choose a high-performance core; there is no low-power option.

Based on this description, it is obvious that greater microarchitectural *diversity* in the set of cores provides the scheduler with greater flexibility. The next section elaborates the intuitive notion of a diverse set of cores.

## 5.3 Obstacles to Diversity

For a heterogeneous processor to provide flexibility at runtime, the set of cores implemented by the processor must be *diverse*. While *diversity* is an intuitively obvious idea, it is difficult to quantify. Ideally, a processor would contain many cores that are tuned to different types of programs and that range from low power to high performance. This would allow the scheduler to maximize performance for all types of programs in all circumstances. However, a maximally diverse processor might contain hundreds of types of cores, while a real processor can only implement a few different core types. In the following, five sub-components of the diversity problem are discussed. First, a generic view of heterogeneous processor design is introduced. If used correctly, this design flow can lead to a diverse set of cores. Second, the problem of selecting cores for diversity is presented. Third, the problem of evaluating the diversity of a set of cores is described. Fourth, the complicating effects of application diversity are discussed. Finally, the issue of unreliable software tools is considered and mitigated.

### 5.3.1 Design Flow

**Figure 5.2** shows a generic design flow for a heterogeneous processor. The design process begins with a design space containing all possible cores. The space is searched for cores that could potentially be implemented—the *candidate set*. This search step is often called *DSE* (design space exploration). Then, a *selection* step is used to determine which cores in the candidate should be implemented on the processor.

**Figure 5.3** illustrates the design flow using the *aes* benchmark and the example design space described in **section 4.3.2** (p. 42). For this example, the design space contains 3000 cores, which are shown using gray points. Most of these cores have such poor power and performance characteristics that they should never be considered for implementation. DSE can be used to search the space for the candidate set of power-performance Pareto-optimal cores, shown with black points. Any core in the candidate set could be implemented, but since there are tens of candidate cores, not all of them can be. A selection step must be used to choose a small number of candidate cores for implementation. One possible selection of four cores is shown using open circles.

If the final set of cores is to be diverse, both the DSE and selection steps must behave correctly. If the DSE step does not find a diverse candidate set of cores, then

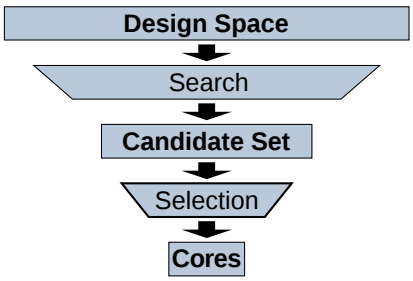


**Figure 5.2:** *Both search (DSE) and selection steps are required to reduce a large design space to a small number of core types that can be implemented on a heterogeneous processor.*

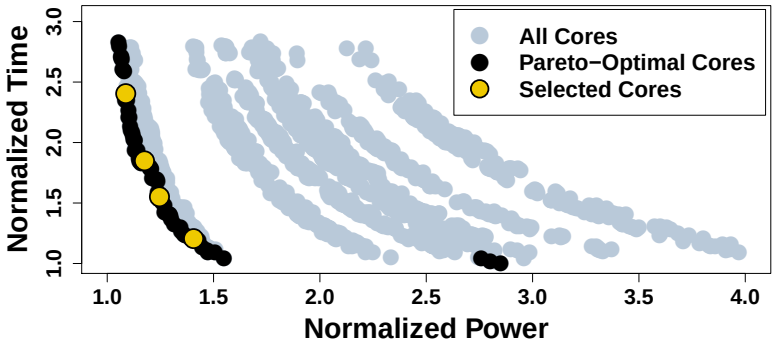


**Figure 5.3:** *All cores, the candidate set (Pareto-optimal cores), and selected cores, shown for the aes benchmark (see **section 4.3.1** (p. 41)). Smaller is better on the axes.*

it is impossible for the the selection step to select a diverse set of cores. Even if the DSE step finds a diverse set of cores, it is still possible for the selection step to choose a set of cores that is not diverse, because the candidate set normally contains many more cores than can be selected. As detailed in **section 3.6.1** (p. 35), there has been substantial research on DSE. DSE research rarely addresses the selection problem, however, the implicit assumption being that a designer with domain knowledge can readily transform DSE results into a selection of cores. The following sections show this assumption to be untrue. Since the focus of this thesis is on selection and not on DSE, random sampling and Pareto-optimality is used instead of a DSE method (see **section 4.3.2** (p. 42)).

### 5.3.2 Selection Problem

Intuitively, the problem of selecting cores is similar to the problem of selecting a team of people for a competition. It is not necessary for each team member to be better at a task than the corresponding member of the opposing team. It is only necessary

that the team is collectively superior to the opposing team. The composition and evaluation of teams is difficult in any context, be it sports or office management. This section considers the problem of selecting a set of cores; the next section considers the evaluation of sets.

A prerequisite to DSE and core selection is a set of benchmarks that is representative of the types of applications that will be run on the final processor. A processor will necessarily be targeted at the types of applications that are used to tune it during design. The flexibility of a heterogeneous processor is maximized when it implements the most diverse set of cores possible—all cores that are Pareto-optimal for at least one representative benchmark. There can, however, be hundreds of such cores. Assuming that DSE is able to find a sufficiently diverse candidate set, enabling flexibility requires the designer to select a small number of cores from the candidate set that preserves the diversity of the candidate set.

The selection problem is non-trivial, because any subset of cores selected from the candidate set must inevitably approximate some aspects of the candidate set, and will inevitably lose some of the diversity of the candidate set. If a selection method is to reduce a candidate set of hundreds of cores down to a number that can reasonably be designed and implemented, it must make value judgments, and must prioritize some features of the candidate set over others. The problem is further complicated by the fact that the effect a given core has on the diversity of a selection is not an intrinsic feature of the core, but is instead dependent on how the core relates to other selected cores—whether a core is selected is dependent on which other cores are selected.

For a selection method to prioritize features of sets of cores, there must exist features that can be prioritized. Current literature recognizes very few such features. A common, first-order feature of heterogeneous processors is *monotonicity* (see **section 3.2.4.1** (p. 22)). A *monotonic* selection of cores is one where the cores have a clear ordering. The set of cores can be ordered by, for example, speed, such that each core is faster than the previous, regardless of which benchmark is considered. That is, a monotonically heterogeneous processor does not attempt to take advantage of variations in benchmark behavior, and it is reasonable to run any benchmark on any core, subject only to runtime requirements for power, speed, etc. A *non-monotonic* selection of cores is one where there is no such ordering. Each core optimizes a metric (e.g., speed) for one or more benchmarks, and each benchmark will perform sub-optimally if run on the wrong core. Monotonic heterogeneity emphasizes runtime flexibility with respect to changing external conditions—sometimes an application

must be run with more or less speed or power, depending on user expectations and the environment (e.g., ambient temperature). Non-monotonic heterogeneity emphasizes runtime flexibility with respect to application behavior—each core will be optimal for different applications. A set of heterogeneous cores can combine aspects of both monotonic and non-monotonic heterogeneity. Based on the accepted definition of monotonicity, monotonic heterogeneity is better suited to provide the flexibility required by mobile devices. However, **section 6.11** (p. 106) and **section 6.12** (p. 110) show that even monotonicity is, in fact, an imprecise blend of two distinct features.

In summary, solving the selection problem requires knowing what the desirable features of a set of cores are. Very few such features exist in current literature. The most common one is monotonicity, but monotonicity is poorly defined.

### 5.3.3 Evaluation Problem

The evaluation problem is closely coupled to the selection problem. For any selection of cores, it must be possible to show whether the selection is better or worse than another selection. If it is not possible to compare two selections, then it is impossible to decide which one should be implemented. It is therefore also impossible to compare and improve upon selection methods.

During the design process, the processor designer could potentially make a small number of intuitive evaluations about sets of cores. However, the design process may require the evaluation of billions of permutations of core types for hundreds of benchmarks. Performing evaluations consistently at this scale requires well-defined metrics. Metrics exist for comparing individual cores and for evaluating system-level throughput (see **section 5.6** below), but there has been little work on comparing sets of heterogeneous cores intended for consumer devices. For example, it is simple to compare cores in terms of speed and power consumption, but if there are two processors, one with a slow and a fast core, and the other with a slightly slower and a slightly faster core, it is not clear what metric will adequately compare the two. The second processor has a higher peak performance and lower minimum power consumption, but the first processor might be more appropriate to an average usage scenario.

### 5.3.4 Program Diversity

The selection problem is complicated by the fact that each program interacts differently with each core's microarchitecture. **Figure 5.4** illustrates this using two hypothetical benchmarks from a processor designer's representative benchmark suite. Each benchmark could be an entire program, a program phase, or a thread from a multi-threaded program. The examples use a trade-off between power and execution time. Power is a hard, physical limit that cannot be exceeded due to thermal and electrical considerations. Execution time should be minimized to increase user satisfaction. **Figure 5.4** (top) shows eight cores, *A*-*H*, that the designer can select from. In this example, the designer has selected a diverse set of cores: *A*, *B*, *D*, and *H*. These cores are Pareto-optimal for benchmark 1—each core is at an optimal power-performance point, ranging from low power to high performance. The selection is flexible for benchmark 1, because the scheduler can run it at a wide range of performance levels.

**Figure 5.4** (bottom) shows the same cores for benchmark 2. For this benchmark, core *H* is no longer Pareto-optimal, since core *D* is both faster and consumes less power. Core *E* is Pareto-optimal, but has not been selected for the processor. Cores *A* and *B* have almost identical behavior. It can be seen that the same set of cores provides much less flexibility to the scheduler when running benchmark 2—benchmark 2 can be run at effectively two power-performance points (*A* or *B*, and *D*).

There are any number of interactions between program behavior and core microarchitecture that could cause a core to be Pareto-optimal for one benchmark but not another. For example, consider the problem of selecting a fast core for a heterogeneous processor using two benchmarks. One benchmark has a large working set, and the other has high ILP. A core with a large data cache will speed up the first benchmark, but the additional, unused cache will only waste power for the second benchmark. A core with a large instruction window will speed up the second benchmark, but the larger instruction window will waste power for the first benchmark. In this case, each core is Pareto-optimal for one benchmark but not the other. A core with both a large data cache and a large instruction window will waste power for both benchmarks, and will not be Pareto-optimal for either. It is not immediately obvious which core the designer should select, as this depends on how many cores can be selected, the relative priorities of the benchmarks, and any other cores that have also been selected. In summary, a set of cores that is diverse for one benchmark is not necessarily diverse for another,

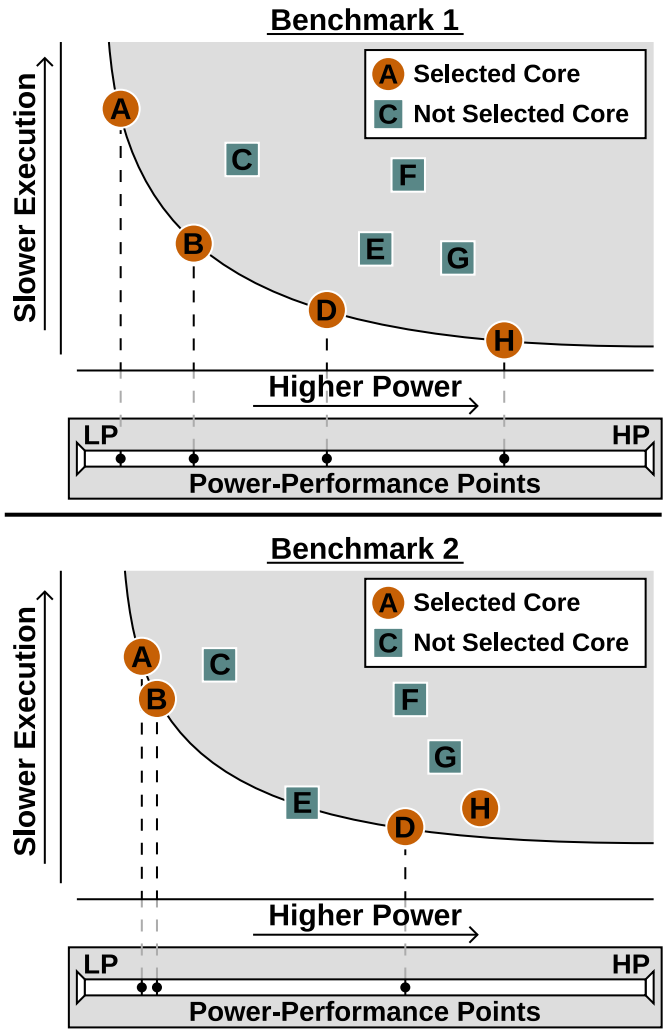


**Figure 5.4:** *(Top) Four selected cores, A, B, D, and H provide a range of power-performance points to benchmark 1, ranging from low power (**LP**) to high performance (**HP**). (Bottom) The same four cores are not as diverse for benchmark 2. H is not even Pareto-optimal. E is Pareto-optimal, but is not selected.*

because flexibility is dependent on the interaction between a core's microarchitecture and a program's behavior.

As noted in **section 3.5.3** (p. 33), the problem of selecting points from a Pareto frontier also appears in the context of elitist genetic algorithms. However, these methods do not apply to the core selection problem. In the case of genetic algorithms, only one Pareto frontier is sampled. In the case of core selection, there is a different Pareto frontier of cores for each benchmark, but one set of cores must be selected to represent all of these frontiers.

### 5.3.5 Tool Unreliability

The final complication of the diversity problem is the reliability of software tools—simulators and power models (see **section 4.2.1** (p. 40) and **section 4.2.2** (p. 40)). If selection and evaluation methods are coupled too closely to a particular set of tools, then they may not work as intended when used with other tools. For example, if a selection method assumes that the possible range of power values for cores is between 250mW and 2W, but the method is used on a range of power values between 1W and 10W, then the selection method might only select cores in the 1W to 2W range. This is an issue of particular concern, as the intended user of selection and evaluation methods is an industrial processor designer, but industrial design tools are generally not available outside industry.

The key to ensuring that selection and evaluation methods are transposable to different modeling ecosystems is to make the assumptions underlying the methods as broad as possible. In the following chapters, the only assumption made regarding tools is that there exists a trade-off between power and performance. This behavior is observed in gem5 and McPAT, is supported by countless studies, and directly follows from first principles: To make a CPU core faster, it must perform more work in parallel. If more work is performed in parallel, more circuits are active, and more power is consumed. **Section 6.2** (p. 75) considers the problem of evaluation reliability in greater detail.

## 5.4 A Power-Constrained Runtime Model

A major roadblock to the development of heterogeneous processors is in analyzing their behavior. The problem is particularly acute for mobile devices, where the CPU shares a power supply and thermal capacitance with a multitude of other components. At any given time, there could be any number of tasks running on the CPU, all competing for access to a limited amount of power. In addition, factors external to the processor can affect how much power the CPU can use. For example, if a smartphone is in a noisy radio environment, power to the antenna may need to be increased, which can reduce power available to the processor. Alternatively, the processor may need to run slower on a very warm day, because there is less thermal headroom to dissipate the heat produced by the processor. The competing priorities and requirements of tasks on the CPU, compounding factors from elsewhere on the device, and factors external

to the device all taken together can make analyzing and optimizing a heterogeneous processor appear to be an intractable problem.

The runtime model suggested here and used in the following chapters solves the intractability of the analysis by eschewing a global view of a mobile device in favor of a task-local view. For a mobile device, the single, first-order requirement is to maintain a power budget. If the power budget is broken, then there is a risk of physical damage due to excessive temperature, excessive electrical current, etc. The power budget for the entire device may vary with time depending on external factors, such as ambient temperature. The second-order requirement for the device is to maximize performance to satisfy the user, while staying within the transient power budget. The device may have other, third-order requirements, but these are ignored for the purposes of the model. Based on these two requirements, the runtime model simply states that when a task is to be run, there is some amount of power available to it, and the execution speed of the task must be maximized without exceeding the available power. Rather than attempt to determine how much power is available to a task at a given time by modeling everything else taking place on the device, the runtime model represents available power as a probability density function (PDF). In one instance when a task is run, there may be a large amount of power available, while in another instance, there may be only little available power. The processor is evaluated with respect to the given task and the probability distribution of available power.

The runtime model is based on the following specific assumptions:

**Assumption 1: Power is limited**

There is a hard limit on the amount of power that the processor can consume.

**Assumption 2: An intelligent scheduler exists**

There is an intelligent scheduler either in the OS or in middleware that controls on which core each task is executed.

**Assumption 3: There is prior knowledge on power**

The scheduler has prior knowledge of the power consumption of each task on each type of core. This could be from off-line profiling or learned at runtime.

**Assumption 4: Available power is measurable**

There are monitoring circuits to determine how much power is available and how much is being used (see, e.g., Liu et al. [113]).

**Assumption 5: Maximum available power per task is capped**

The processor is designed to accommodate the most power-hungry task-core combination. The scheduler will never give a task more power than this.

**Assumption 6: Task migration between cores can be ignored**

Short tasks finish quickly and do not benefit from migration. Long-running programs can be treated as a series of tasks, where the scheduling of each task is determined independently of the previous one.

**Assumption 7: The required core is always available**

Dark silicon predicts that processors will have many more cores than can be powered simultaneously [50]. The runtime model assumes that if there is enough power available for a task to use a given type of core, then an instance of that core is free and can be used.

The runtime model requires a scheduler that allocates power to each task based on factors such as the task's priority, the amount of power available to the processor, the temperature of the processor, remaining charge in any batteries, etc. When such a scheduler is assumed, then the performance of any given task can be analyzed based solely on the power PDF—the amount of power that the task is likely to receive. This allows the designer to analyze a processor on a benchmark by benchmark basis rather than attempting a global analysis.

## 5.5 Intuitive Diversity Considerations

The discussion in the preceding sections leads to an intuitive conception of diversity. A diverse selection of cores should have good *spread*—the cores should enable both low-power and high-performance operation. The selection should also have *uniformity*—the cores between the extremes should be evenly spaced. Furthermore, the selection should have good spread and uniformity for all benchmarks in the representative set. While spread and uniformity are intuitively simple, they are difficult to optimize objectively, and the following will show that there are non-trivial problems associated with achieving each.

### 5.5.1 Spread

*Spread* simply describes how far the lowest-power and highest-performance cores are from each other. For an example, see the power-performance point scale in **figure 5.4**. If the amount of power the scheduler can make available to a task is less than the power required to run the task on any core, then the task cannot run. Therefore, extending the spread of cores in the low-power direction increases the likelihood that the processor will be available to do work. Similarly, if the scheduler has large amounts of power available, it must have access to high-performance cores to convert the available power into user satisfaction. Extending the spread of cores in the high-performance direction increases peak performance. Given these observations, the immediate intuition may be to maximize diversity by maximizing spread. However, this will not guarantee a good selection of cores. At the low-power and high-performance extremes of the Pareto-optimal set of cores, tiny gains in one metric are made at the expense of large losses in the other. For example, re-architecting the lowest-power core to consume 1% more power may lead to a 10% increase in performance, and re-architecting the fastest core to slow it down by 1% may lead to a 10% reduction in power. While this reduces spread, it may nevertheless be desirable: The slight power increase at the low-power end might have an insignificant effect on the processor's availability, while noticeably improving performance when power is tightly constrained. The slight performance reduction at the high-performance end might have an insignificant effect on performance, while noticeably increasing the chance that the fastest core can be used. It can be concluded that while a broad spread of cores is necessary for diversity, maximizing spread is not guaranteed to be a good design decision.

Achieving spread is complicated by diversity in program behavior. As already noted in **section 5.3.4**, increasing the size of a given microarchitectural structure in a core can increase the performance of the core for one benchmark, but it might only increase the power consumption of another benchmark. A selection of cores should provide a fair spread of power-performance points to all benchmarks in the representative suite, but this leads to a combinatorially complex problem where increasing spread for one benchmark can decrease it for another.

### 5.5.2 Uniformity

A set of cores enables flexibility if the cores are uniformly distributed in the design space. Cores that have similar behavior, like the clustered set in **figure 5.1**, limit

flexibility at runtime since the scheduler can only choose from a small range of power and performance levels. While *uniformity* is an intuitively obvious concept, it is difficult to define. Existing uniformity metrics depend on carefully defined constants, and it is therefore difficult to determine whether the metrics are measuring uniformity or their own tuning parameters (see **section 5.6.3** (p. 68)). Even if uniformity can be defined precisely, uniformity alone does not guarantee a diverse selection of cores. If spread is poor, then diversity will be poor regardless of uniformity. Even if spread is good, uniformity can be poor. Similarly to spread, cores that are uniformly spaced for one benchmark may be clustered together for another. For example, if there are several cores with varying instruction window sizes, then for a benchmark with high ILP, the cores implement different power-performance points. In contrast, for a benchmark with low ILP, the cores have similar performance, but some will consume more power and will not be Pareto-optimal.

## 5.6 Limitations of Current Metrics

If microarchitectural diversity were measurable, then methods for improving diversity could be developed. Metrics used to evaluate heterogeneous processors must take into account the diversity of cores and the consequent flexibility at runtime. Without a diversity metric, it is impossible to objectively compare selections of cores, and it is therefore impossible to compare the efficacy of competing selection methods. Simple measurement techniques, such as evaluating power and performance, are not sufficiently descriptive for heterogeneous processors, since heterogeneous processors contain many types of cores that operate at many power and performance levels. The most common metrics currently used to evaluate heterogeneous processors fall into two general categories: throughput metrics and summary metrics. Metrics in these two categories are inadequate for evaluating heterogeneous processors, because the features they measure are (sometimes subtly) different from features that are desirable in heterogeneous processors. A third category of metrics—diversity metrics—comes from research into multiobjective evolutionary algorithms (MOEAs). While the diversity preservation components of MOEAs have an intent similar to core selection for diversity, core selection places stricter requirements on metrics that MOEA metrics cannot meet. The limitations of the three categories of metrics are detailed below.

### 5.6.1 Throughput Metrics

There has been significant research into metrics for evaluating the throughput of processors. This area of research became relevant when processors gained the ability to run multiple threads in parallel, either using SMT (simultaneous multi-threading) or by implementing multiple cores. Throughput metrics are used to evaluate the total work completed by a processor. The metrics are sometimes coupled with fairness metrics to ensure that increases in throughput are fairly distributed among threads, and sometimes with turnaround time metrics to evaluate the average execution time of tasks (see **section 3.5.2** (p. 33)). Since heterogeneous processors are also multicore processors, they are sometimes evaluated with the same throughput metrics used for homogeneous multicores.

Throughput metrics are compelling, because the concept of total work completed is intuitively obvious. However, throughput is only rarely useful for evaluating heterogeneous processors in mobile devices. The user of a mobile device is not interested in maximizing the number of instructions executed by several parallel threads in, e.g., a 24-hour period. Most of the cores on a mobile device will be idle most of the time, and when the cores are used, the user requires responsiveness and a short turnaround time for tasks, even if these lower the overall throughput of the processor. Van Craeynest and Eeckhout [182], for example, observe a trade-off between system-level throughput and the turnaround time of individual tasks, which suggests that optimizing mobile devices for throughput produces the opposite of the desired effect.

Throughput and other similar metrics are based on average execution speed, which hides variations in speed. If a task runs very slowly for a minute, and then very quickly for another minute, then the throughput may be acceptable, but the user is likely to be unhappy with the uneven performance. To avoid this averaging behavior, the runtime model (**section 5.4**) assumes that a task has exclusive access to a CPU core, ensuring that the task has constant performance. The metrics proposed in **chapter 6** primarily operate on a task-level granularity, and there is therefore no way to hide uneven performance. (Note that a task is not necessarily an entire program, but may be a program phase.)

For the avoidance of doubt, throughput metrics are valid for evaluating throughput-oriented, fully subscribed processors where turnaround time is of secondary importance. Mobile processors simply do not fit into this category.

### 5.6.2 Summary Metrics

As described in **section 2.2.3** (p. 10), a *summary metric* is a metric that combines two or more other metrics into a single figure of merit. A summary metric can potentially help manage a trade-off between metrics, as different combinations of values for the underlying metrics can have the same overall figure of merit. There are obvious parallels between summary metrics and the selection problem. A solution to the selection problem requires a set of cores that implement different trade-offs between power and performance. If a summary metric could combine power and performance such that a slow, low-power core; a fast, high-power core; and cores at various points in between have the same value for the metric, then the summary metric could be used to select at least a preliminary set of cores from a candidate set. Each selected core would have a different power and performance, but they would all be equally "good" with respect to the summary metric.

The most common summary metric used for evaluating processors is $ED^2$—energy-delay-squared product. $ED^2$ is equivalent to $PD^3$ (power-delay-cubed product), and is inversely proportional to $IPS^3/W$ (instructions-per-second-cubed per Watt). I.e., $ED^2$ evaluates a trade-off between power and performance. In the following, it is first shown that $ED^2$ cannot be used to select heterogeneous cores for diversity. It will then be argued that summary metrics in general fail to select for diversity.

#### 5.6.2.1 $ED^2$ Limitations

The $ED^2$ metric is a measure of *efficiency*—it takes into account both the speed of execution and the associated energy cost. A recurring theme in microarchitecture research is that a CPU core should be energy efficient—regardless of its speed, the core should maximize the amount of useful work performed with the energy that is consumed. An example of this is found in Zyuban and Kogge [198], who coin the term *energy-efficient architecture*. $ED^2$ was originally proposed by Martin et al. [121] as a voltage-invariant method of evaluating circuits in an asynchronous processor. While the energy and delay of a circuit are both dependent on voltage, the $ED^2$ of the circuit is not. $ED^2$ has since been used to evaluate all types of processors (see **section 3.5.1** (p. 32)).

For a compute-bound workload running on a core with DVFS, $ED^2$ can be expected to be roughly constant across all DVFS levels. This can lead the designer of a heterogeneous processor to assume that $ED^2$ should also be constant for a diverse set

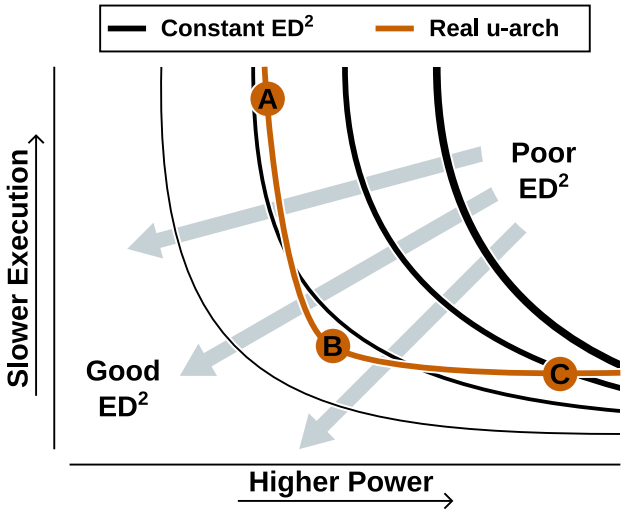


**Figure 5.5:** *The black lines show constant energy efficiency ($ED^2$) from low power to high performance. Ideally, low-power and high-performance cores would have equal efficiency, but this is difficult to achieve in practice.*

of heterogeneous cores. One core might consume very little energy, but would be slow. Another core might be fast, but would consume large amounts of energy. Together, the cores would span a diverse set of power-performance points.

Selecting cores with similar, optimal $ED^2$ values is, however, an unreliable selections methodology, as it is unlikely that different microarchitectures with different power-performance characteristics can actually be implemented with equally good $ED^2$. This has been shown quantitatively by Azizi et al. [7]; an intuitive explanation is given in **figure 5.5**. Each black line in the figure is a constant $ED^2$ value. *A*, *B*, and *C* are cores. *B* corresponds to an in-order or simple out-of-order core. The high efficiency (work per unit of energy) of *B* makes it a good fit for battery-operated devices and throughput-oriented data centers. *C* corresponds to a wide, out-of-order core with aggressive prefetching and speculation. Cores like *C* are used for applications where latency is more important than efficiency (e.g., gaming). Finally, *A* is an extremely low-power core. There are fewer examples of such cores in the market. Cores like *A* might be used when strict thermal limits are in place, and low-power operation is more important than speed or efficiency.

If cores *A*, *B*, and *C* in **figure 5.5** could be designed to have equal efficiency, then $ED^2$-optimization could be used to make at least a preliminary selection of diverse cores. However, the energy spent on aggressive speculation in *C* makes it inevitable that some energy will be wasted and $ED^2$ will be worse than for *B*. Similarly, the long

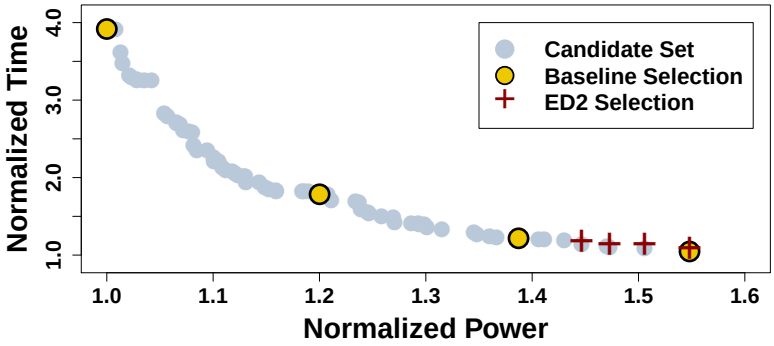


**Figure 5.6:** *The $ED^2$-optimized selection has a favorable mean normalized $ED^2$ (1.1), but the cores do not give good coverage of the design space. The* Baseline *selection contains both low-power and high-performance cores and covers the design space, but its mean normalized $ED^2$ is poor (10.0). The candidate set is shown for reference. The data is for the* aes *benchmark.*

execution time of $A$ gives it a lower efficiency than $B$. More broadly, the $ED^2$ metric is premised on the observation that the speed of a circuit increases linearly with voltage, and the energy consumption of a circuit increases with the square of voltage. To apply $ED^2$ to a space of heterogeneous cores is to assume that microarchitectural changes that increase speed increase energy consumption by the square of the speed increase. However, microarchitectural changes alter the structure of circuits, rather than the applied voltage. A CPU core contains a large number of microarchitectural structures that are all interdependent, but have different functions, different performance characteristics, and different interactions with workloads. It is therefore obvious that the squared relationship between speed and energy cannot be assumed to hold, and that the $ED^2$ metric cannot be used to select cores for diversity. Intuitively, as illustrated by **figure 5.1**, a processor with cores like $A$, $B$, and $C$ is more diverse than a processor with only cores like $B$. However, average $ED^2$ is optimized by a processor with only type $B$ cores.

**Figure 5.6** shows the limitations of the $ED^2$ metric using data from the example dataset (see **section 4.3** (p. 41)). The gray points make up the candidate set—all the power-performance Pareto-optimal cores in the space. Two sets of cores are manually selected from the candidate set. The cores in the *Baseline* selection are selected to emphasize diversity, and the cores in the $ED^2$ selection are selected to minimize $ED^2$. The *Baseline* selection contains a broad spread of cores, ranging from low-power to high-performance. This selection can run tasks quickly when there is a large amount of power available to the processor, and it can continue running tasks on the slower

cores even when power is tightly constrained. For this particular design space, $ED^2$ is minimized at the high-power end of the candidate set. It is clear from inspection that the $ED^2$-optimized selection does not offer the same flexibility as the *Baseline* selection. If power is limited even slightly, then none of the cores in the $ED^2$ selection can be used.

Despite the obvious limitations of the $ED^2$ selection, the average normalized $ED^2$ of the four cores in the $ED^2$ selection is only 1.1—10% worse than the best $ED^2$ possible in the design space. The average $ED^2$ for the cores in the *Baseline* selection is 10.0. This is 900pp (percentage points) worse than the best possible, or 9× worse than the $ED^2$ selection. Again, this demonstrates that the features that make for good aggregate $ED^2$ efficiency are likely to result in a poor set of heterogeneous cores.

#### 5.6.2.2 General Limitations

The previous section argued that an $ED^2$-optimization approach cannot lead to a diverse selection of cores, because $ED^2$ is minimized at one point in the design space, but a diverse set of cores must span a range of points in the design space. While $ED^2$ and the inversely proportional $IPS^3/W$ are some of the most widely used summary metrics in computer architecture, other metrics of the general form $E^iD^j$ have also been used. However, adjusting the weights of the E and D terms simply shifts the point in the design space where the metric is minimized; it does not introduce a spread of points. These limitations of summary metrics have previously been studied in the context of MOEAs (see, e.g. Kursawe [100], Zitzler and Thiele [196]), and apply equally to using summary metrics to select diverse cores.

Some authors consider using a set of $E^iD^j$-type summary metrics to capture a spread of points across the design space, where each pair of *i*-*j* values emphasizes a different point in the space [3]. This also does not help solve the selection or evaluation problems; it simply transforms the problem of selecting cores into a problem of selecting pairs of *i* and *j* values. The more general limitation of $E^iD^j$ metrics, as noted by Azizi et al. [7], is that all pairs of *i* and *j* values are arbitrary—there is no good reason to think that any given trade-off between *i* and *j* is better than another. There are even situations where energy and delay are directly correlated and cannot be traded off (see **section 3.5.1** (p. 32)).

The fundamental problem with summary metrics is as follows: A summary metric assumes a fixed exchange rate between the underlying metrics—in the case of $ED^2$, speed is worth twice as much as energy. The real microarchitectural design space

is unlikely to follow any fixed exchange rate, and a summary metric will therefore preclude diversity by valuing one point in the design space more highly than others. However, if a designer wishes to include some regions of the design space and exclude others, this should be done explicitly, rather than implicitly through summary metrics. Even if a summary metric could be devised such that it exchanges the underlying metrics in exactly the same way as the real microarchitectural design space, it would not help solve the selection and evaluation problems. A summary metric that perfectly matches the design space would hide diversity. All selections of cores, whether diverse or clustered, would have the same average score with respect to the summary metric.

Ultimately, when selecting cores, the designer should ensure that the candidate set of cores is Pareto-optimal for power and performance and that the selected cores are appropriately spaced in the design space. $E^iD^j$-type metrics cannot contribute to either of these goals. It is obvious why considerations like power, energy, speed, and throughput are potentially interesting to a designer or user, but it is not at all clear what additional information summary metrics contribute or why they should be optimized.

### 5.6.3 Diversity Metrics

Heterogeneous core selection can be simplistically viewed as a type of multiobjective optimization problem—the set of selected cores must optimize the inversely proportional quantities of power and execution time. Elitist MOEAs keep a running list of top (*elite*) solutions. The set of elite solutions is of finite size and must provide coverage of the entire Pareto-optimal solution set (see, e.g., Zitzler and Thiele [196]). The problem of selecting some elite solutions to represent the solution set is similar to the problem of selecting some cores from a candidate set.

Despite the apparent similarity, selecting elite solutions is a fundamentally different problem from selecting cores; while techniques used to evaluate elite solutions can inspire techniques for evaluating selections of cores, they cannot be guaranteed to be usable in their original form for evaluating heterogeneous sets of cores. This is because in general, selecting elite solutions is a simpler problem than selecting cores, and because specific metrics that have been developed for use in MOEAs have limitations that make them difficult to apply to sets of cores. Selecting elite solutions is the simpler problem for three reasons: First, as already noted in **section 5.3.4**, one selection of cores must approximate many, different Pareto-optimal frontiers, one per benchmark. A set of elite solutions need only sample one Pareto frontier. Second, elite solutions

exist only to guide an algorithm, while selected cores will be implemented. There is therefore a greater correctness requirement for core selection methods. Third, the number of elite solutions can be substantially larger than the number of selected cores. 100 elite solutions is common [40]; current heterogeneous processors implement two types of cores [69]. Core selection must achieve much more with far fewer data points.

In addition to the above general differences, various diversity metrics that have been used with MOEAs have features that make them difficult to use consistently for evaluating sets of cores. The most common such feature is that many diversity metrics require *tuning*—they are dependent on constants that must be set correctly for the metric to function. The tuning requirement makes it difficult to interpret metrics and compare studies. This issue is discussed further in **section 6.2** (p. 75) and **section 6.6.5** (p. 88), and is also raised by Deb et al. [40].

## 5.7 Limitations of Current Selection Techniques

The central benefit of single-ISA heterogeneity is that it is able to shift some design-time decisions to runtime. A designer strategically selects a set of cores to implement so as to enable flexibility later. The final decision regarding which core a task is run on is made at runtime with the benefit of runtime information. Despite the advantages of heterogeneity, there has been next to no research into strategies that the designer can employ to select a flexible set of cores. The following sections discuss four selection methods and their limitations. The first two are naïve approaches, and are included for completeness. The final two have been developed for server processors, and are not applicable to mobile processors. Multiobjective optimization techniques are not discussed, as they do not address the selection problem (see **section 3.6.1** (p. 35)).

### 5.7.1 Max Selection

*Max* selection is a strategy that selects cores to maximize a single metric, like performance or efficiency. It can potentially lead to as many selected cores as there are benchmarks used to tune the processor, as a different core might maximize the metric for each benchmark. Lee and Brooks [107] use Max selection, and then use clustering to reduce the number of core types down to the required number. The primary limitation of a Max selection strategy is that it can only lead to non-monotonic heterogeneity, and it therefore cannot provide flexibility between low power and high

performance execution. It is also not clear whether and under what circumstances can it be justified that many cores are used to optimize just one metric. Lee and Brooks [107] observe average $ED^2$ improvements of up to 30% when using cores selected for $ED^2$ with a Max selection strategy instead of the best single core. However, performing Max selection for $ED^2$ for the EEMBC benchmarks using the infrastructure in **section 4.3** (p. 41) leads to an insignificant $ED^2$ improvement of 4% over the single, best core.

### 5.7.2 Max-Budget Selection

*Max-Budget* selection is a natural extension of Max selection that adds a budget for some resource. Resources that can be budgeted include power and silicon area—a Max-Budget selection strategy might seek to maximize performance within a given power budget, for example.

The underlying limitation of a Max-Budget strategy is that hard limits on resources are external to the design space and are imposed on the design space without insight into the features of the space. As such, hard limits can lead to poor trade-offs. An extreme example is a design space with two competing cores, where one core consumes 50% more power and is 0.1% faster than the other. If performance is to be optimized, and if the faster core fits in the power budget, then the faster core will be selected despite the fact that trading 50% in power for 0.1% speed is generally unreasonable. The converse is also possible: There could exist a core that both requires marginally more power than allowed for by the budget and is significantly faster than any core within the budget. A Max-Budget selection strategy would exclude such a core even though a designer could make allowances for it.

Similarly to Max selection, Max-Budget selection can only select non-monotonic cores. It may be suggested that different power budgets can be used to select cores at a range of power-performance points. However, this approach suffers from the same problem described above—an unfavorable core could be just within one of the power budgets, and a favorable core could be just above a budget. Furthermore, while the highest power budget could be determined from physical power or thermal limits, the other budgets will be arbitrary.

### 5.7.3 GA Selection

One of the more sophisticated core selection strategies is by Navada et al. [134], who use a genetic algorithm (GA) to intelligently perform both design space exploration

and core selection in one step. This *GA* selection strategy is a form of Max selection and Max-Budget selection. The genetic algorithm performs two functions: It searches the design space, and it handles the combinatorial complexity of selecting a few cores for many benchmarks. The genetic algorithm can either attempt to find the fastest set of cores, or the fastest set of cores under a power budget. GA selection is evaluated in **chapter 8**.

While the GA selection strategy solves the complexity problem of the Max-Budget selection strategy, it still only selects a non-monotonic set of cores. As such, the cores do not offer flexibility between low power and high performance, and are only appropriate for high-performance computers, not mobile devices. The GA could be used to select cores under a range of power budgets, but as already noted in the previous section, these power budgets will inevitably be arbitrary.

### 5.7.4 Clustering Selection

The state of the art in heterogeneous core selection is the *Clustering* selection strategy by Guevara et al. [71]. The Clustering strategy selects cores from a candidate set by grouping similar cores together using k-means clustering, and then selecting a representative core from each cluster. This leads to low-power cores, high-performance cores, and cores in between, as required by mobile processors. The processor designer defines how many different types of cores are needed by setting the number of clusters that k-means should produce.

The Clustering strategy is evaluated in **chapter 8**. It's main limitation is that it attempts to minimize runtime risk in server processors. While the Clustering strategy is relevant to mobile devices, its conservative nature prevents it from taking advantage of some of the specialization possible with heterogeneous processors.

## 5.8 Summary

Bridging the gap between design space exploration and a heterogeneous processor requires solutions to the selection and evaluation problems. DSE methods produce candidate sets of cores, but they offer no means of choosing cores from candidate sets. The key requirement for a selection of cores is microarchitectural diversity that leads to an acceptable level of runtime flexibility for all relevant types of programs. Existing

selection methods cannot provide this diversity, and existing evaluation methods cannot measure it.

# Chapter 6:

# Metrics for Sets of Heterogeneous Cores

Heterogeneous processors enable flexible execution under variable power budgets through microarchitectural diversity. Traditional, homogeneous processors require one, well-designed CPU core that can be instantiated many times. Heterogeneous processors require many, well-designed cores, with the additional requirement that the cores also perform well together. The need for collective optimization of cores is a marked complication over the pattern established by homogeneous processors. This chapter introduces a suite of quantitative metrics for evaluating a range of features exhibited by sets of heterogeneous CPU cores. These metrics lay the groundwork for rigorous heterogeneous processor design, evaluation, and analysis.

## 6.1 Introduction

A significant component of heterogeneous processor design is the evaluation problem, as described in **section 5.3.3** (p. 55). To design a heterogeneous processor, it must be possible to quantify features of the processor rigorously and at scale. It would be unreasonable to base design decisions solely on a processor designer's intuition. The CPU core selection problem (**section 5.3.2** (p. 53)) is combinatorially complex, with a nearly infinite number of potential core combinations that could be implemented. Even if a human designer could consistently offer unbiased evaluations of sets of cores, the sheer number of evaluations required by the design process demands well-defined metrics so that evaluation can be automated. Cost-benefit analyses are also dependent on rigorous metrics. Each design decision has a monetary design and fabrication cost.

Without quantifying a decision's effects, it cannot be determined whether the decision is a correct one for the processor.

When evaluating heterogeneous processors, it is crucial to be precise about what features, exactly, are being evaluated. It is easy to conflate two related, but distinct concepts, and it is easy to define summary metrics that hide the true features of processors. For example, a recent work [182] performs design space exploration (DSE) to optimize a heterogeneous processor for STP (system throughput) and ANTT (average normalized turnaround time) (see **section 3.5.2** (p. 33)). The authors find that Pareto-optimal points in the trade-off between the two metrics can be achieved with just two core types—i.e., they find that only minimal heterogeneity is required. This may lead a reader to conclude that minimal heterogeneity is sufficient in all circumstances. However, because the chosen metrics operate on the system level, it is not possible to determine *why* the system cannot take advantage of more than two types of cores. It may be that the memory bandwidth and area budget used in the experiments inhibited heterogeneity; it may be that the candidate set of cores used to compose the processor was too small; it may be that when STP and ANTT are the optimization targets, then two core types will always be sufficient; it may even be that two core types are universally sufficient for all heterogeneous processors; or it may be due to a completely different reason. The system-level metrics are simply not descriptive enough to draw general conclusions. Furthermore, while system-level metrics are applicable to heavily utilized server processors and similar problems, they are inadequate for under-subscribed, mobile, consumer devices. The average throughput of a smartphone, for example, is completely irrelevant if the throughput swings wildly between low and high values—the user will be displeased by the performance variation even if average performance is acceptable.

This chapter contributes to the solution of the evaluation problem by defining eight new metrics for evaluating *sets* of *heterogeneous* cores. I.e., the metrics evaluate the behavior of a selection of cores collectively rather than the features of any single core, and the metrics consider the types of cores, not the number of each type. These eight metrics are not necessarily exhaustive—there may be other, as yet unidentified, features of sets of cores that could be optimized. The metrics are independent—a designer can choose not to use some of the metrics if they quantify features he is not interested in. Crucially, the metrics are tools that provide the designer with information and insight to aid in making intelligent design decisions; they are not interchangeable quantities that can simply be traded off via a cost model. The metrics

are not intended to evaluate considerations such as *uncore* components (e.g., networks-on-chip or NoCs), whole systems, scheduling algorithms, or even the business case for heterogeneity. If a designer finds benefits from, e.g., eight core types, but then finds that the NoC hides these benefits, then this should motivate NoC research. Similarly, if an operating system designer struggles to schedule to several core types, this demonstrates the need for better scheduling algorithms. The engineering cost of many different cores might be prohibitive in a given market, but this can change quickly as markets change and design automation improves. A designer must understand the effects of each system component to be able to draw valid conclusions about system design. These eight metrics evaluate one part of the system—the set of cores—thereby enabling a more rigorous approach to the selection and evaluation problems. The metrics are summarized in **table 6.1**.

The remainder of this chapter is arranged as follows: The second section describes the motivating goals that guide the form that the metrics take. The third section introduces the assumptions that underlie the metrics. The fourth section considers some basic metrics—naïve metrics that are included for completeness and to support later discussion. The following seven sections each discuss one of the eight metrics, except for the section on uniformity, which contains two metrics. Each metric is introduced with an intuitive description, followed by a numerical definition and example use cases. The metrics are all premised on the runtime model described in **section 5.4** (p. 58). The final section considers the implications of the runtime model's available power PDF on the formulation of the metrics.

## 6.2 Motivating Goals

As described above, the overall goal of the metrics is to provide a set of tools that can be used to aid the design of heterogeneous processors. Beyond this general theme, there are four principles that guide the way in which metrics should be defined. These four principles, or goals, are listed here and described below.

**Goal 1:** Avoid constants and tuning

**Goal 2:** Use relative instead of absolute values

**Goal 3:** Have no dependence on a baseline architecture

**Goal 4:** Have an intuitive interpretation

| Name | Feature | Optimization | Per-Benchmark / All Benchmarks | Optimal Cores / All Cores | Summary Function |
|---|---|---|---|---|---|
| **KS Test** | Standard statistical technique. Used here to measure uniformity | Smaller-is-better | Per-benchmark | Optimal cores | Arithmetic mean |
| **Localized Non-Uniformity** | Uniformity | Smaller-is-better | Per-benchmark | Optimal cores | Arithmetic mean |
| **Gap Overhead** | Slowdown over ideal set with all cores | Smaller-is-better | Per-benchmark | Optimal cores | Arithmetic mean |
| **Set Overhead** | Slowdown over a different set of cores | Smaller-is-better | Per-benchmark | Optimal cores | Arithmetic mean |
| **Availability** | Likelihood that the set of cores can be used | Larger-is-better | Per-benchmark | Optimal cores | Arithmetic mean |
| **Effective Speed** | Average speed of a set of cores | Larger-is-better | Per-benchmark | Optimal cores | Harmonic mean |
| **Generality** | Amount of per-benchmark specialization in a set of cores | N/A | All benchmarks | All cores | N/A |
| **Monotonicity** | Dependence of core ordering on benchmark | N/A | All benchmarks | All cores | N/A |

**Table 6.1:** *Summary of metrics. Metrics applied on a per-benchmark basis can be summarized for a set of cores using a summary function. Per-benchmark metrics are only applied to cores in the selection that are Pareto-optimal for the benchmark.*

**Goal 1:** If one must choose values for constants or perform some other tuning on a metric, then one can never be certain whether the metric is actually evaluating the processor, or whether it is only evaluating its own tuning. This severely limits the ease of use and applicability of a metric. The user of the metric must prove that constants have been given correct values, even though there is no single, well-defined definition for "correct." This, in turn, makes comparing results from two different studies difficult, because both must tune the metric similarly. Tunable constants in metrics also make it easier for users of the metric to doctor results. Finally, since metrics are used during design as well as for evaluation, a poorly tuned metric can direct the design process in the wrong direction.

**Goal 2:** Processor design is reliant on simulators and power models, since design precedes implementation. Modeling tools that are fast enough to be used for design space exploration can generally be assumed to produce output that is accurate in relative terms, but not in absolute terms. E.g., if the size of a microarchitectural structure is doubled, then the relative power for the core can be expected to track appropriately (as reported by a tool like McPAT). The absolute power in Watts is implementation-dependent, however, and is difficult to predict across a design space. Goal 2 states that metrics should not rely on the absolute correctness of a number from a tool, but should only assume relative correctness. As a consequence, metrics that meet this goal can only report relative results—a metric can report a 10% improvement in execution speed, for example, but not a 10 second improvement in speed.

**Goal 3:** Goal 3 follows from goal 2. For metrics to be relative, they must compare against some baseline. Works on homogeneous processors generally use a baseline architecture. In the heterogeneous case, it is unclear what a baseline architecture should be, as there is no accepted, generic, heterogeneous processor, and there are a nearly infinite number of possible ways of composing a heterogeneous processor. The metrics in this chapter and the algorithms in **chapter 7** use the whole design space as a baseline, and report quantities relative to the design space. For example, the power consumption of a benchmark on a core is reported relative to the power of the given benchmark when running on its lowest-power core. The definitions later in this chapter will further clarify this approach. **Section 7.4** (p. 121) revisits this goal in the context of a selection algorithm.

**Goal 4:** Finally, goal 4 states that metrics should be intuitive to understand. While the formulation of a metric may be involved, it should be easy to understand what the resultant number represents. As systems become more complicated, greater effort

must be made to ensure that humans can continue to reason about the processors being designed and to sanity-check the algorithms used during design.

## 6.3 Assumptions

A small number of assumptions underlie the metrics defined in this chapter. The most significant of these is the runtime model and its associated assumptions, described in **section 5.4** (p. 58). While many of the runtime model's assumption are not satisfied by current mobile devices, all of them are within reach of current or near-future technology.

The runtime model assumes a scheduler that schedules tasks based on an available power probability density function (PDF). The metrics in this chapter are therefore dependent on an intelligent scheduler that has knowledge of power PDFs. While this is a non-trivial dependency, it means that the metrics inherit the versatility of the runtime model. For example, if a task is a single-threaded, high-priority application, then the scheduler will tend to make more power available to it, and the evaluation will be with respect to a PDF that is skewed toward high power. Conversely, a task might be just one thread in a multi-threaded application. In this case, the scheduler can only make small amounts of power available to the task, since the task always runs concurrently with other tasks. Evaluation will be with respect to a PDF that is skewed toward low power. For simplicity, the metric definitions in this chapter assume a flat PDF—all values of available power between the minimum usable power and maximum possible power are equally likely. Real PDFs could be determined empirically or by modeling usage scenarios, for example. **Section 6.13** describes applying the metrics to cases where the PDF is not flat.

Finally, as noted in **section 6.1**, the metrics consider the selection of *cores*. The metrics therefore assume that uncore components can be designed to fully take advantage of the heterogeneity provided by the cores.

## 6.4 Example Data

Each of the metrics in this chapter is demonstrated using examples. For consistency, the demonstrations make use of the same four selections of cores where possible. The cores are manually chosen from the example dataset described in **section 4.3.2** (p. 42). The complete list of cores in the design space is in **appendix A**. The *Baseline* selection

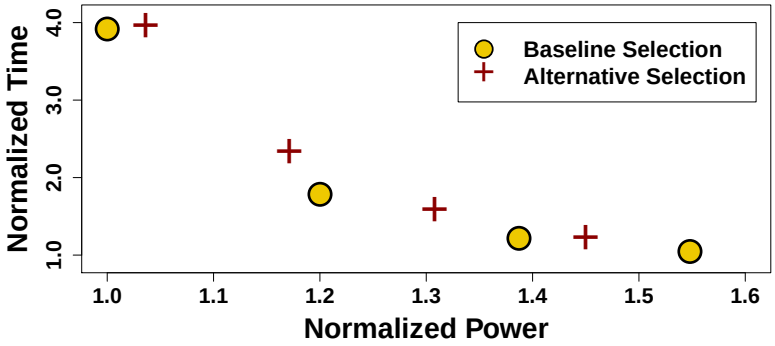


**Figure 6.1:** *The* Baseline *and* Alternative *selections, shown for* $bm_1$.

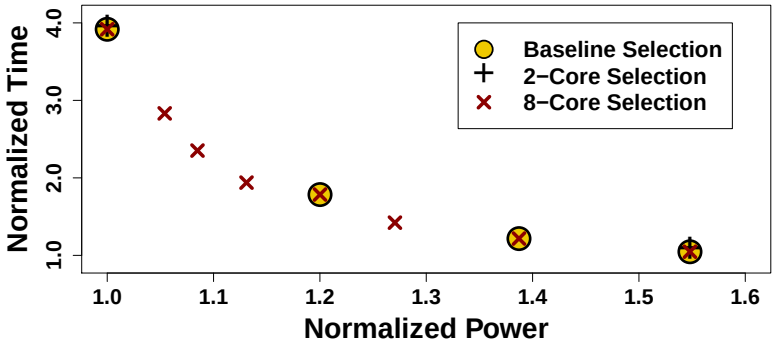


**Figure 6.2:** *The* Baseline *selection, and 2- and* 8-Core *selections, shown for* $bm_1$.

is intended to be a reasonably good set of four heterogeneous cores. It contains cores #60, #671, #1731, and #2367. The *Alternative* selection of cores is similar to, but intuitively worse than the *Baseline* selection. It contains cores #417, #713, #1065, and #2027. The *Baseline* and *Alternative* selections are used with metrics that compare different selections that contain the same number of cores each. The *2-Core* selection contains only two cores—#60 and #671 from the *Baseline* selection. The *8-Core* selection contains all four cores from the *Baseline* selection, as well as cores #597, #1546, #2524, and #2804. The *2-Core*, *Baseline*, and *8-Core* selections are used with metrics that evaluate the effects of increasing the number of cores in a selection.

Rather than use all benchmarks from **section 4.3.1** (p. 41), this chapter illustrates metrics with five benchmarks from the EEMBC suite: *aes*, *cjpeg*, *huffde*, *mpeg4encode*, *rgb2cmyk*. For brevity, these are referred to as $bm_{1\text{-}5}$. **Figure 6.1** shows the *Baseline* and *Alternative* selections for $bm_1$. **Figure 6.2** shows the *2-Core*, *Baseline*, and *8-Core* selections for $bm_1$. Figures in this chapter plot cores by power and execution time, normalized to the best power and best execution time for the given benchmark by any core in the design space.

## 6.5 Basic Metrics

This section describes four naïve metrics: minimum, maximum, spread, and minimum or maximum with budget. There is nothing intrinsically incorrect about these metrics, nor is there anything particularly novel about them. The metrics quantify obvious features of a selection of cores and can be useful during the design process. However, due to their simplicity, these metrics have very little descriptive power, and cannot be used to guide the selection of cores. The metrics have a close correspondence to some of the existing selection strategies described in **section 5.7** (p. 69).

### 6.5.1 Minimum

The minimum (min) of a physical quantity for a selection of cores is simply the minimum of that quantity achievable with that selection. For example, a designer may be interested in determining what the minimum possible execution time is for a benchmark when given a set of cores. A particularly important feature of a selection of cores is its minimum operating power. The available power PDF and its dependent metrics are only useful when there is enough power to use at least one core. If the amount of power available to a task is less than the minimum operating power of the set of cores, then the task cannot be run, and little further analysis is possible (see the availability metric, **section 6.9**, and the PDF discussion, **section 6.13**).

### 6.5.2 Maximum

The maximum (max) of a physical quantity for a selection of cores is the maximum of that quantity achievable with that selection. For example, the maximum amount of power that a task can consume when running on a selection of cores has consequences on thermal design and power delivery.

### 6.5.3 Spread

The spread of a given physical quantity for a selection of cores is simply the range between the minimum and maximum. As described in **section 5.5.1** (p. 61), spread is a necessary but insufficient condition for enabling runtime flexibility. Spread is important, because a designer may well wish to fix the minimum and maximum acceptable levels of power consumption, and only select and evaluate cores in the fixed range.

### 6.5.4 Budgeted Minimum and Maximum

Budget-based metrics evaluate the minimum or maximum of one physical quantity when another physical quantity is constrained. For example, a designer may wish to find the minimum execution time of a task on a set of cores given a maximum power budget, or he may wish to find the minimum power consumption of a task given a maximum time budget (i.e., a deadline). As described in **section 5.7.2** (p. 70) and **section 5.7.3** (p. 70), a budgeted metric can be a sufficient design criterion for homogeneous processors and some server-oriented heterogeneous processors, but is inadequate for evaluating the runtime flexibility demanded by mobile processors.

## 6.6 Uniformity

The first two metrics quantify *uniformity*—how evenly the selected cores cover the candidate set. **Section 5.5** (p. 60) argued that a heterogeneous mobile processor must be diverse to enable runtime flexibility. Diversity is maximized when the selected cores provide uniform coverage of the candidate set. The candidate set may be clipped to a range of power values (the required spread). An intuitive description of a uniformity metric is presented first. Uniformity can be evaluated using either the KS test or the localized non-uniformity metric. The two approaches are described in turn and then compared. It is argued that the localized non-uniformity method is more robust. Finally, the localized non-uniformity metric is compared to various diversity metrics.

### 6.6.1 Intuition

**Figure 6.3** shows the *Baseline* selection of four power-performance Pareto-optimal cores described in **section 6.4**. Assume that the processor designer has the option of adding a fifth core to the selection: either core #2416 or core #1188. The two options for the fifth cores are plotted with crossed circles, and the complete candidate set of Pareto-optimal cores is shown in gray for reference. The cores are plotted for $bm_1$. The axes are normalized power and execution time. 1.0 on the power axis is the minimum power consumed by any core in the design space when executing $bm_1$. 2.0 on the power axis is twice that amount. Since normalization involves a division, the values have no units. The normalization procedure is identical for the time axis.

Intuitively, there are two reasons to select core #1188 over core #2416. The first is flexibility at runtime. Core #1188 roughly bisects the gap between core #2367

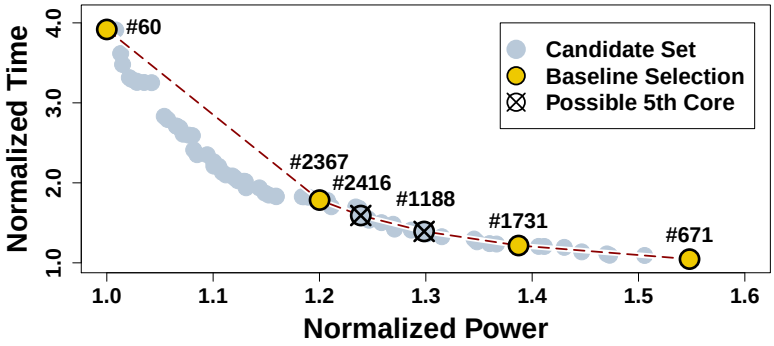


**Figure 6.3:** *Non-uniformity helps determine that core #1188 is a better addition to the* Baseline *selection than core #2416 when considering benchmark $bm_1$. The dashed line shows Euclidean distance between cores.*

and core #1731. The runtime scheduler can trade off power and performance in regular increments, leading to a smoother user experience. In contrast, there is a large performance drop between core #1731 and core #2416, and only a small drop between core #2416 and core #2367. The second reason is engineering effort. Core #2416 almost duplicates core #2367. If core #2416 is implemented, then there is only a low likelihood that there will be enough power for core #2367 but not enough for core #2416. Core #2367 will be nearly redundant.

The intuition, then, is that a metric is required for measuring whether cores are uniformly spaced along the Pareto-optimal frontier, or whether the cores appear in clusters. The metric should consider worst-case behavior, such as the gap between core #60 and core #2367. It cannot, however, be dominated by worst-case behavior, because then it cannot help select between core #2416 and core #1188.

### 6.6.2 Kolmogorov-Smirnov Test

The two two-sided, two-sample Kolmogorov-Smirnov test (KS test) is a statistical technique used to determine whether two sets of sampled points are drawn from different, unknown distributions [36, p.456]. The following sections describe and demonstrate the use of the KS test to quantify uniformity.

#### 6.6.2.1 Application to Uniformity

The KS test reports the $T_1$ metric. $T_1$ ranges from 0.0 to 1.0. If $T_1$ is close to 0.0, then it is very likely that two sets of sampled points are drawn from the same underlying distribution. The process of evaluating uniformity with $T_1$ is as follows: $T_1$ is evaluated

on a per-benchmark basis using power values. For this metric, it does not matter whether the power values are normalized. The first set of sampled points contains those cores from the candidate set that are Pareto-optimal for the given benchmark. The second set of sampled points contains the cores from the selection that are Pareto-optimal for the benchmark. Since the test is only applied to Pareto-optimal points, results are identical when using time instead of power.

In this application, it is already known that the two sets of sampled points come from the same underlying distribution—a distribution determined by the microarchitectural design space of cores. Calculating $T_1$ allows one to determine how much the selection of cores "looks like" the candidate set—how representative the selected cores are. A representative selection captures most of the diversity of the Pareto-optimal set, while a selection that is not representative is not diverse. An advantage of the KS test is that it favors selections that cover the entire range of cores even if the candidate set contains clusters of cores that skew the distribution.

#### 6.6.2.2 Example: Evaluating Diversity

A small-scale example of the KS test is shown in **figure 6.4** and **table 6.2**. The example contains a small candidate set composed of eight types of cores. This set is different from the *8-Core* selection described in **section 6.4** above, and has been manually selected from the complete candidate set to illustrate the KS test. **Figure 6.4** plots the eight cores for three benchmarks, $bm_{1\text{-}3}$. The cores' locations in the power-time space vary depending on the benchmark as a result of behavioral differences among benchmarks (see **section 5.3.4** (p. 56)). Two sets of three cores each are manually selected from the small candidate set. The *Clustered* selection looks intuitively poor—all cores are toward the high-performance end of the space, and there is little diversity. The *Diverse* selection looks intuitively good—the cores are more evenly spaced and cover the full range of the candidate set. The *Diverse* and *Clustered* selections are better and worse selections, but not necessarily the best and worst selections of three cores from the small candidate set of eight.

**Table 6.2** shows $T_1$ for both selections, and demonstrates that the $T_1$ metric corresponds to intuition. $T_1$ between the candidate set and itself ("All Cores") is obviously 0.0. $T_1$ for the *Clustered* selection is noticeably worse than $T_1$ for the *Diverse* selection. It can be seen that it is possible to use $T_1$ to report the diversity of a selection of cores.

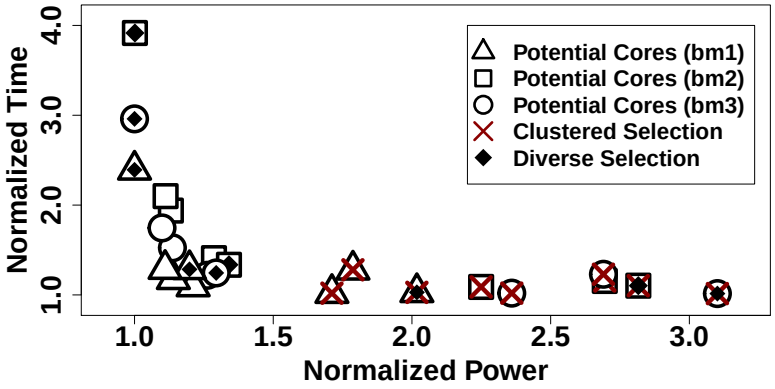


**Figure 6.4:** *Eight cores with increasing cache sizes and instruction windows, shown for $bm_{1\text{-}3}$. Some selections of three cores enable flexibility; others do not.*

| Selection | $T_1$ | | |
|---|---|---|---|
| | $bm_1$ | $bm_2$ | $bm_3$ |
| All Cores | 0.0 | 0.0 | 0.0 |
| *Diverse* | 0.33 | 0.33 | 0.33 |
| *Clustered* | 0.83 | 0.67 | 0.67 |

**Table 6.2:** *$T_1$ values for selections of cores in **figure 6.4** (smaller is better)*

### 6.6.3 Localized Non-Uniformity

*Localized non-uniformity* measures how well a selection of cores covers a candidate set, and is similar in intent to the application of the KS test described above. Informally, it can be said to measure "clumpiness"—how clustered the selection of cores is. The following provides the mathematical definition for the metric and demonstrates one possible use case with an example.

#### 6.6.3.1 Definition

Localized non-uniformity is measured over a 1-dimensional distribution of points on a per-benchmark basis using the selected cores that are Pareto-optimal for the given benchmark. The metric begins with the coordinates of the points in 2-dimensional, normalized, unit-less space. I.e., the power and time axes are normalized to the lowest possible power and lowest possible execution time for that benchmark. The 2-dimensional space is flattened to one dimension using Euclidean distance—the points are ordered by one of the axes, and the first point becomes the 1D origin. Since the points are Pareto-optimal, it does not matter which axis is chosen for ordering. The Euclidean

distance between the first and second point is measured. This distance becomes the 1D offset for the second point. The Euclidean distance in the 2-dimensional space between the second and third point becomes the 1D distance between the second and third points, and the process continues until all points have been flattened. The dashed lines in **figure 6.3** show the Euclidean distance between cores. Localized non-uniformity is represented by the Hebrew כ (*kaph*) as a mnemonic for the *c* in "clumpiness." It is defined in **equation 6.1**.

$$כ_{bm} = \frac{(u_1 - R_{min}) + (R_{max} - u_N) + \sum_{i=2}^{N-1} d_i}{R_{max} - R_{min}} \qquad d_i = \left| u_i - \frac{u_{i-1} + u_{i+1}}{2} \right| \tag{6.1}$$

$כ_{bm}$ is localized non-uniformity for benchmark *bm*.

$N$ is the number of points.

$u_i$ is the 1-dimensional coordinate of point $i$, where $i$ is in the range $[1, N]$.

$R_{min}$ is the lower-bound of the range over which כ is calculated.

$R_{max}$ is the upper-bound of the range.

$d_i$ is the distance from point $i$ to the midpoint between its two neighbors.

The value of $כ_{bm}$ is guaranteed to fall in the range $[0, 1]$, where 0.0 is a perfectly uniform distribution, and 1.0 is a single, tight cluster of points. The range that $כ_{bm}$ is calculated over is defined separately from the points, as $u_1$ and $u_N$ are not necessarily the smallest and largest possible values for $u$. I.e., the first point to be flattened is the 2-dimensional lower-bound of the range over which כ is calculated. This point, when flattened, becomes $R_{min}$. The last point to be flattened is the upper-bound of the range, and becomes $R_{max}$. $R_{min}$ and $R_{max}$ will often correspond to the lowest-power and highest-performance cores for *bm* in the candidate set. If these cores are also part of the selection, then $R_{min}$ is equal to $u_1$, and $R_{max}$ is equal to $u_N$.

#### 6.6.3.2 Discussion

As the name suggests, localized non-uniformity measures how far points are from being uniformly distributed, but it does so by taking into account only a point's neighbors and not the entire distribution. **Equation 6.1** calculates how far the first point is from the start of the range and how far the last point is from the end. For every other point, it calculates how far it is from being uniformly distributed with respect to its two neighbors (how far it is from the midpoint). The most noteworthy aspect of this formulation is that it is easy to normalize—the sum of all the distances will never

be greater than the size of the full range. The formulation also ensures translation-invariance—a set of points will continue to have the same ב value even if it is shifted within the range $[R_{min}, R_{max}]$. Finally, whether points appear clustered or uniformly distributed depends on how large the range $[R_{min}, R_{max}]$ is, and **equation 6.1** reflects this.

Localized non-uniformity is most useful for ranking different selections of the same number of core types in preparation for further analysis. For example, taking the four cores plus core #2416 in **figure 6.3** gives $ב = 0.39$. If core #1188 is used instead of core #2416, $ב = 0.33$.

#### 6.6.3.3 Example: Identifying Redundancy

**Section 6.6.1** above argued that closely clustered cores are redundant—there is little benefit to designing two different cores with almost identical power and performance. The localized non-uniformity metric, ב, can be used to quickly identify the worst cases of redundancy. As an example, **figure 6.5** shows ב for the *Baseline* selection and the *Alternative* selection for all five benchmarks. ב is different for each selection-benchmark combination, as each benchmark interacts differently with each core's microarchitecture. The *Baseline* selection has a ב less than 0.33 for all benchmarks. This gives the designer confidence that if the *Baseline* selection is sufficiently uniform for, e.g., $bm_1$, then it is sufficiently uniform for other benchmarks as well. (See **figure 6.1** for the *Baseline* and *Alternative* selections plotted for $bm_1$.)

The *Alternative* selection has better uniformity than the *Baseline* selection for $bm_1$, $bm_2$, and $bm_4$. However, ב for $bm_3$ and $bm_5$ on the *Alternative* selection is particularly poor—0.42 and 0.38, respectively. The designer will wish to investigate why non-uniformity is so high for these benchmarks, and whether the use of the *Alternative* selection in a processor is justified. **Figure 6.6** shows the *Baseline* and *Alternative* selections for $bm_3$. The reason for the high non-uniformity value is obvious: The two fastest cores in the *Alternative* selection have almost identical behavior, and $bm_3$ is therefore not receiving the full flexibility benefits of heterogeneity. Ideally, with four different cores, a workload could be run at four different power-performance points, but the *Alternative* selection can run $bm_3$ at only three power-performance points. This result can help the processor designer choose between the *Baseline* and *Alternative* selections. It may be the case that $bm_3$ must have access to four distinct power-performance points, or it may be that the processor's priorities are elsewhere and the *Alternative* selection is sufficient for $bm_3$.

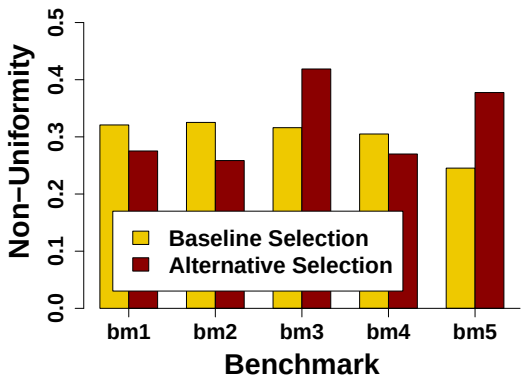


**Figure 6.5:** *Localized non-uniformity, ↄ, shows that clustering behavior is worst for benchmark $bm_3$ when running on the* Alternative *selection of cores.*

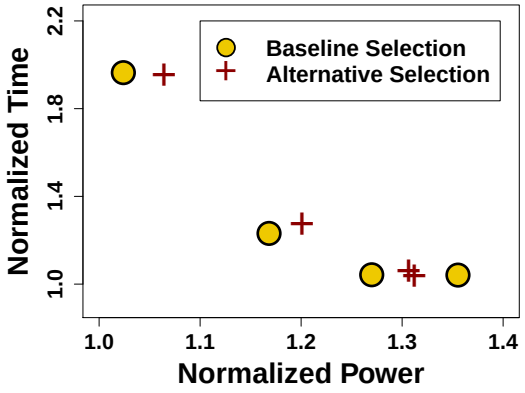


**Figure 6.6:** Baseline *and* Alternative *selections for benchmark $bm_3$. The two fastest cores in the* Alternative *set have almost identical behavior.*

Given only two sets of cores and only five benchmarks, a designer could find the worst-case clustering of cores simply by visual inspection. However, as the number of selections and benchmarks increases, a human designer will quickly be overwhelmed, while the localized non-uniformity metric can easily scale to any number of benchmarks, any number of cores in a set, and any number of sets.

### 6.6.4 KS Test and Localized Non-Uniformity

The KS test and localized non-uniformity are very similar in intent, though substantially different in formulation. While the KS test is an accepted statistical technique, it has a significant shortcoming that makes localized non-uniformity the better of the two metrics for evaluating selection of cores. Specifically, the KS test is a function of the largest gap between two points (cores). In the example in **figure 6.3** for example, the KS test evaluates the selection based solely on the distance between core #60 and core #2367. $T_1$ is 0.35 regardless of whether core #2416 or core #1188 is added. In contrast, the localized non-uniformity metric evaluates all cores with respect to their neighbors, and all distances between cores have an effect on ↄ. This is helpful for a designer, because the designer may have good reasons for maintaining a large gap between cores. In the case of **figure 6.3**, $T_1$ can only be improved by adding a core between core #60 and core #2367. However, it might be the case that a new core between core #60 and core #2367 is physically or financially infeasible, and the designer's only options are core #2416 or core #1188.

### 6.6.5 Localized Non-Uniformity and Related Metrics

There are a number of techniques that aim to quantify concepts similar to the כ metric. As noted in **section 5.6.3** (p. 68), many of these metrics require tuning, and therefore fail to meet the first goal for metrics (**section 6.2**). In contrast, neither the KS test nor localized non-uniformity contain tunable constants.

In addition to the tuning problem, these metrics have other shortcomings. The ε-coverage error and δ-uniformity metrics by Sayın [156] are both dependent on min and max operators. As a result, they are liable to the same masking problem as the KS test (see **section 6.6.4** above). כ has a further advantage over the ε and δ pair in that כ is only one quantity and therefore much easier to optimize. The entropy-based diversity metric by Farhang-Mehr and Azarm [56] requires the user to tune a grid size and a density function. The metric is also not normalized and does not have a well-defined range of values, which makes it difficult to use in comparisons. Δ-nonuniformity, defined by Deb et al. [40], is most similar to localized non-uniformity. If כ is a "local" metric, then Δ could be considered a "global" metric. However, Δ is not normalized and can exceed 1.0 for extreme distributions. As demonstrated by the example in **section 6.6.3.3**, stability under extreme conditions is perhaps the most important feature of a metric used to evaluate selections of cores.

Uniformity has similarities with "clusterability"—how readily can points be placed into clusters. Ackerman and Ben-David [1] evaluate "clusterability" by first clustering points, and then measuring the quality of the clusters. Given that clustering is an NP-hard problem, this is an exceptionally inefficient approach. It should not be necessary to find the cluster that each data point belongs to only to determine whether the data even contains clusters. Since evaluating the uniformity of a selection of cores does not require finding specific clusters, the localized non-uniformity metric is both an efficient and a sufficient tool for the processor designer.

## 6.7 Gap Overhead

*Gap overhead* is a metric for quantifying how wasteful a selection of heterogeneous cores is on average. The uniformity metrics in the previous section measured how regular the gaps between selected cores are. Gap overhead measures the average effect of the gaps between cores. The intuition behind the metric is described first, followed by its definition, a discussion, and an example of its use.

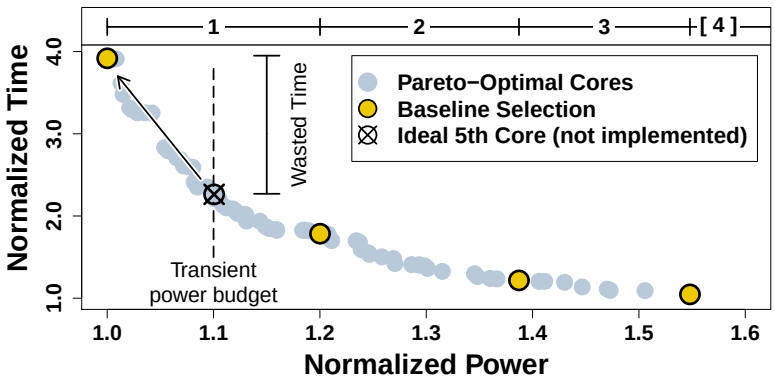


**Figure 6.7:** *If the amount of runtime power available to $bm_1$ is at the dashed vertical line, then execution must take place on the slowest core. The difference in execution time between the ideal, unimplemented core and the slowest core is overhead. Intervals (i in **equation 6.2**) are shown along the top.*

### 6.7.1 Intuition

One of the primary motivators for heterogeneity is the ability to select an appropriate power-performance point for a task at runtime. If a task has a high priority, then it can be allocated a large amount of power so that it can run quickly. Often, however, it will be the case that only a limited amount of power is available to a task, since there may be other tasks running, the task might have a low priority, or the processor may be close to its thermal limit, etc. In such cases, it is desirable to run the task as quickly as possible without exceeding the amount of power that the scheduler is willing to make available to it. It is highly unlikely that a heterogeneous processor contains a core that can execute the given task using exactly the amount of power available. In most cases, the task must drop down to a slightly slower core, which incurs a slowdown *in addition* to the slowdown created by the power limit. Gap overhead quantifies this additional slowdown.

**Figure 6.7** illustrates gap overhead with $bm_1$ and the *Baseline* selection of four cores. In this particular instance, it so happens that there is a small amount of power available to $bm_1$, the *transient power budget*. The ideal core for $bm_1$ (the crossed circle) is not implemented, and execution must drop to a lower-power core, as shown with the arrow. As a result, $bm_1$ runs slower than it ideally would, thereby wasting time. Gap overhead averages this wasted time for all possible transient power budgets.

### 6.7.2 Definition

Gap overhead (GO) is measured on a per-benchmark basis using the selected cores that are Pareto-optimal for the given benchmark. **Equation 6.2** gives the definition of GO. GO is based on the intervals between cores, as shown in **figure 6.7**. It is evaluated with respect to a pair of *resources*—physical quantities that should be minimized. The *X-resource* is the *constrained* resource. At a given time, the constrained resource cannot exceed a given value. The *Y-resource* is the *wasted* resource. The effect of the gaps between cores is quantified as overhead with respect to the $Y$-resource. In this particular application, the constrained resource is power, as it represents a limit that cannot be exceeded. The wasted resource is time, as the gaps between cores lead to a slowdown. It is conceivable, however, that GO could be used with other pairs of measured quantities (e.g., power and energy).

$$\mathrm{GO}_{bm} = \sum_{i=1}^{N} \alpha_i \left( \frac{\max(Y_i) - \overline{Y_i}}{\overline{Y_i}} \right) \qquad \alpha_i = \frac{\max(X_i) - \min(X_i)}{X_{max} - X_{min}} \tag{6.2}$$

$\mathrm{GO}_{bm}$ is gap overhead for benchmark *bm*.

$X$ is the constrained resource (power in this case).

$Y$ is the wasted resource (time in this case).

$\min(X_i)$ is the minimum value of $X$ in interval $i$. I.e., it is the power of the left-bound core of $i$.

$\max(X_i)$ is the maximum value of $X$ in interval $i$. I.e., it is the power of the right-bound core of $i$.

$\max(Y_i)$ is the maximum value of $Y$ in interval $i$. I.e., it is the time of the left-bound core of $i$.

$\alpha_i$ is a weighting factor for interval $i$.

$X_{min}$ is equivalent to $\min(X_1)$—the lower bound of the range over which gap overhead is calculated.

$X_{max}$ is the upper bound of the range over which gap overhead is calculated. See below for details.

$\overline{Y_i}$ is an estimate for the average of resource $Y$ in interval $[\min(X_i), \max(X_i)]$ (or $[\min(X_N), X_{max}]$ for the final interval). See below for details.

$N$ is the number of intervals between the core types that are Pareto-optimal for *bm*, plus a final interval $[X_N, X_{max}]$.

$GO_{bm}$ calculates how much of the $Y$-resource is wasted in each interval in the average case, and takes a weighted average across all intervals. It is calculated using absolute values for $X$ and $Y$ ($X$ and $Y$ are not normalized). $GO_{bm}$ uses only those cores in the selection that are Pareto-optimal for $bm$. The division by $\overline{Y_i}$ removes the units and makes the overhead relative (goal 2, **section 6.2**). $\overline{Y_i}$ is an estimate for the average value of the wasted resource in interval $i$. It is here defined as the arithmetic mean of the $Y$ values of all known Pareto-optimal cores in interval $i$, even if they are not part of the set of selected cores. $\overline{Y_i}$ could also be determined by extrapolation or from first principles, but care must be taken to avoid compromising goal 1. $\alpha_i$ provides a weight for each interval. The range over which GO is calculated is $[X_1, X_{max}]$. $X_{max}$ is the maximum amount of power that will ever be available. It is here defined as the maximum power consumed by any benchmark on the most powerful selected core, but a designer will likely be able to use known physical limits instead.

### 6.7.3 Discussion

Gap overhead measures how much extra time execution takes on average because all possible cores cannot be implemented. It helps answer the question, "How much would the average case benefit if another core were added; i.e., should another core be added?" For the example in **figure 6.7**, gap overhead is 0.25—assuming that $bm_1$ can happen to have any amount of power available to it within the valid range, and assuming all power values are equally likely, then on average, $bm_1$ will take 25% longer to execute than it would in the ideal case. If the fifth core shown in **figure 6.7** were also implemented, gap overhead would drop to 0.15. By measuring gap overhead, a designer can determine the average benefit of adding another core type, and whether the engineering effort of another core can be justified.

One should be aware of two considerations when using gap overhead: First, when evaluating the effects of adding cores, the minimum power core must stay constant. Otherwise, the range over which gap overhead is calculated changes, breaking comparability. Second, gap overhead measures the average case, and is therefore not as sensitive to extremes as localized non-uniformity. For example, returning to **figure 6.3**, gap overhead is 0.26 regardless of whether $A$ or $B$ is chosen. Gap overhead and non-uniformity should be used together: GO works best when comparing different numbers of cores; $\supset$ works best when comparing different selections of the same number of cores.

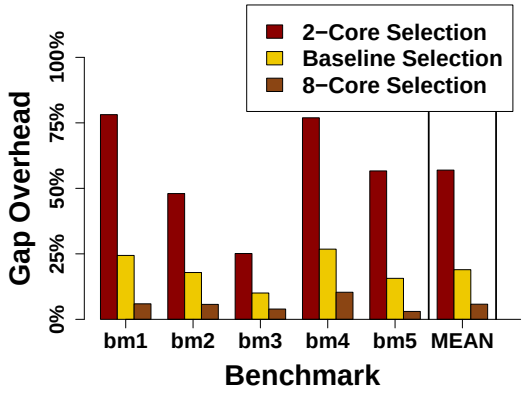


**Figure 6.8:** *When moving from two to four to eight cores (**figure 6.2**), the average amount of time wasted (gap overhead) decreases from 58% to 19% to 6%.*

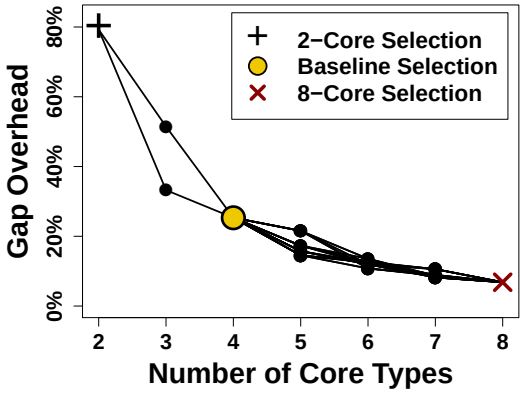


**Figure 6.9:** *Adding Pareto-optimal core types always reduces gap overhead (GO), but some core types reduce overhead more than others. GO is shown for $bm_1$.*

### 6.7.4 Example: Adding Core Types

Gap overhead, GO, measures how much more time execution takes because a processor cannot implement cores at all possible power-performance points to match all possible transient power budgets at runtime. GO can be used to determine when to stop adding cores—if an extra core would only marginally reduce gap overhead, then the designer may decide that the engineering effort required to implement the core outweighs its benefits. As an example, **figure 6.8** shows gap overhead for the *2-Core*, *4-Core* (*Baseline*), and *8-Core* selections from **figure 6.2**. With two cores, gap overhead is quite high—execution of $bm_1$ and $bm_4$ takes nearly 80% longer than theoretically possible. Moving to four and then eight cores, GO drops significantly, and average GO at eight cores is only 6%. The benefit of using four core types rather than two is obvious, but there are diminishing returns to using eight types rather than four. It is up to the designer to determine when the effort to engineer an additional core is no longer justified by a reduced GO. It is expected that the designer would use a weighted average of GO values, and would give benchmarks that represent high-priority or frequently executed tasks a higher weight, as the time wasted by these tasks is more important.

GO can also be used to determine which cores to add to a set and in what order. **Figure 6.9** shows GO for the *2-*, *4-*, and *8-Core* selections, as well as for intermediate numbers of cores. For example, there are two paths between the *2-Core* and *4-Core*

selections, depending on the order that cores are added to the *2-Core* set. One of these paths is clearly better. One 3-core selection has a GO of 51%, while for the other, GO is 33%. Similarly, for five cores, GO ranges from 15% to 22%. This information is useful to a designer for two reasons: First, the designer may simply wish to use GO to select a core to add to a set. Second, the designer may have already decided to implement, e.g., a 4-core processor, but wishes to ship a 3-core processor as an intermediate product while the fourth core is being finalized. GO helps determine which core best complements an existing set and should therefore be designed first.

## 6.8 Set Overhead

*Set overhead* extends gap overhead to measure how much slower one set of heterogeneous cores is compared to another. Conversely, set overhead can also be interpreted as speedup—how much faster the faster set of cores is. The intuition behind the metric is described first, and the definition is then presented. An extended discussion details considerations that should be made when using set overhead. The section finishes with two example use cases of set overhead.

### 6.8.1 Intuition

Gap overhead from the previous section quantifies how much of a resource, such as time, is wasted because a heterogeneous processor cannot implement all possible cores. Set overhead extends this concept to compare the cores on two (potentially completely) different heterogeneous processors. Assume, for example, two competing algorithms that both select four core types to implement. The first algorithm picks the cores in the *Baseline* selection, and the second algorithm makes an *Alternative* selection (see **section 6.4**). When a task is to be run, the probabilistic runtime model (**section 5.4** (p. 58)) will make some amount of power available to the task. Since the two selections contain different cores, one will run the task slower—i.e., with more time wasted—than the other. This quantity is *set overhead*.

**Figure 6.10** demonstrates set overhead. For the demonstration, it is assumed that the hypothetical algorithm that made the *Alternative* selection was based on a DSE method that was not able to find cores as close to the Pareto frontier as the algorithm used for the *Baseline* selection. The amount of power available to $bm_1$ at a given time is shown by the dashed line. The ideal, but unimplemented core for the given

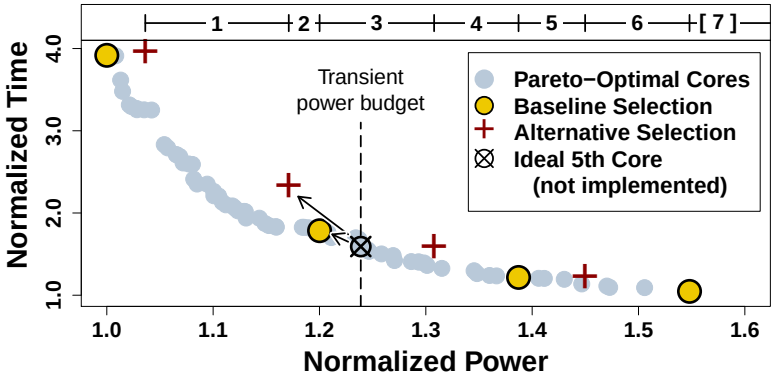


**Figure 6.10:** *Set overhead compares two selections of cores. If the amount of runtime power available to $bm_1$ is at the dashed vertical line, execution will take place on the cores indicated by the arrows. For the power budget shown, the* Alternative *selection wastes more time than the* Baseline *selection. Intervals for* ***equation 6.3*** *are shown along the top.*

power budget is shown using the crossed circle. From inspection, it can be seen that in this case, the core in the *Alternative* selection is slower, and the *Alternative* selection wastes more time than the *Baseline* selection. *Set overhead* quantifies this additional waste across the range of transient power budgets that is common between the two selections.

### 6.8.2 Definition

Set overhead (SO) is measured on a per-benchmark basis for two competing sets of cores. From each set, only the cores that are Pareto-optimal for the given benchmark are used. Set overhead is defined similarly to gap overhead, and makes the same assumptions about constrained and wasted resources (see **section 6.7.2**). SO replaces the comparison to the complete Pareto-optimal set used in GO with a comparison to another selection of cores.

$$\mathrm{SO}_{bm} = \sum_{i=1}^{N} \alpha_i \left( \frac{\max(Y_{A,i}) - \max(Y_{B,i})}{\overline{Y_i}} \right) \qquad \alpha_i = \frac{\max(X_i) - \min(X_i)}{X_{max} - X_{min}} \tag{6.3}$$

| | |
|---|---|
| $\mathrm{SO}_{bm}$ | is set overhead for benchmark $bm$. |
| $X$ | is the constrained resource (power in this case). |
| $Y$ | is the wasted resource (time in this case). |
| $\min(X_i)$ | is the minimum value of $X$ in interval $i$. |

| | |
|---|---|
| $\max(X_i)$ | is the maximum value of $X$ in interval $i$. |
| $\max(Y_{S,i})$ | is the maximum value of resource $Y$ for selection $S$ for interval $i$. |
| $S$ | is a selection of cores (in this case, either the *Baseline* selection $B$, or the *Alternative* selection $A$). |
| $\alpha_i$ | is a weighting factor for interval $i$. |
| $X_{min}$ | is the lower bound of the range over which set overhead is calculated. It is given by $\max(X_{B,1}, X_{A,1})$. |
| $X_{max}$ | is the upper bound of the range over which set overhead is calculated. |
| $\overline{Y_i}$ | is an estimate for average $Y$ in interval $i$. |
| $N$ | is the number of intervals between cores, taking both sets into account. See below for details. |

Like gap overhead, $SO_{bm}$ uses absolute (not normalized) $X$ and $Y$ values, and is calculated for each interval between cores, where the intervals are weighted using $\alpha$ and then summed. The definitions of $\min(X_i)$, $\max(X_i)$, $X_{max}$, and $\overline{Y_i}$ are also identical. The differences are in how the intervals are defined, the definition of $X_{min}$, and the overhead calculation before the multiplication with $\alpha$. For set overhead, the intervals are calculated with respect to all core types, regardless of which selection a core belongs to. If the minimum power values for the two selections $B$ and $A$ are not identical, the first interval is discarded (see **figure 6.10**). In this interval, only one processor can run a workload, and a comparison is therefore undefined. The subtraction $(\max(Y_{A,i}) - \max(Y_{B,i}))$ evaluates for each interval how much more time one selection of cores takes compared to the other. This is then expressed relative to the ideal time estimate, $\overline{Y_i}$. Since intervals are defined by two sets of cores, either $\max(Y_{B,i})$ or $\max(Y_{A,i})$ will always fall outside $i$. In **figure 6.10**, for example, if available power is in $i = 3$, then the usable core from the *Alternative* selection ($A$) is the lower-bound of interval $i = 2$.

### 6.8.3 Discussion

The relationship between set overhead and speedup, the differences between set overhead and gap overhead, and the range of power values over which set overhead is calculated all require further discussion.

#### 6.8.3.1 Set Overhead and Speedup

Set overhead measures *slowdown*—if set overhead is 0.50, for example, then the slower selection of cores requires, on average, 50% longer to complete a task. Set overhead can also be interpreted as *speedup*. If SO $= 0.50$ and a task takes 1.0 seconds on the faster selection, then it will take 1.5 seconds on the slower selection. The faster selection is $1.5\times$ faster $(1.5/1.0)$, or has a speedup of 50% $((1.5/1.0-1.0)\times 100)$. The faster selection will require 33% less time to complete $((1.0-1.0/1.5)\times 100)$ the task.

#### 6.8.3.2 Comparison to Gap Overhead

It is important to note that set overhead is not just a difference of gap overhead values. Set overhead compares two selections, whereas gap overhead compares a selection to the limit set containing all Pareto-optimal cores. For the example in **figure 6.10**, the *Baseline* GO is 0.25, the *Alternative* GO is 0.26, but set overhead is 0.03—in the average case, the *Alternative* selection is 3% slower than the *Baseline* for $bm_1$. From inspection, one might expect SO to be larger. However, the *Alternative* selection compensates for its slower cores with better coverage of the lower half of the power range, and the speed difference between the sets is negligible in interval 1.

Strictly speaking, gap overhead is a subset of set overhead where on of the sets contains all Pareto-optimal cores. There are four reasons for defining GO and SO separately: The first reason is that the questions, "Should a core be added?" (GO), and "Is one set better than another?" (SO) are substantially different in nature, even though the metrics use similar methods for answering both. Gap overhead measures how close a set of cores is to the ideal (but unrealistic) limit. Set overhead measures how close one reasonable set of cores is to another.

The second reason is the frequency of use cases. It is expected that a designer will frequently need to determine the benefits of adding one more core (GO), but some designers may never need to determine if a (potentially completely) different set of cores is better (SO). It is therefore reasonable to define a simpler and more specific metric for the more common problem, and keep the more complicated and more general metric for the rarer problem.

The third reason is interpretability. Using set overhead to measure the difference between, e.g., a four-core and a five-core set provides less information than doing the same with gap overhead, because gap overhead is more closely related to the ideal

limit. For example, set overhead for a 4-core and a 5-core set may be 10%, but it is not clear whether this is significant or how much room there is for further improvement. If, however, adding the fifth core reduces gap overhead from 100% to 93%, then it is clear that the core makes very little difference, and there is still plenty of room to improve. If adding the core reduces gap overhead from 12% to 1%, then it is a very effective core, and there is no further room to improve.

Finally, it is possible that the designer will define $\overline{Y_i}$ differently for GO and SO, and keeping the two metrics separate helps avoid confusion. An example of using GO and SO together with differently derived $\overline{Y_i}$ values is in **section 6.8.5** below.

#### 6.8.3.3 Power Range Considerations

When interpreting SO values, the designer must remember that SO is evaluated only over the common power range. If the lowest-power core in each selection has a different power consumption (if $X_{A,1} \neq X_{B,1}$), then there is a region of power values where only one processor can execute a benchmark, and a comparison between selections in this region is undefined. If this were not the case, then at low levels of available power, one set of cores would complete a task in finite time, while the other set would require infinite time. SO would also be infinite. The processor that can execute in the undefined interval is somewhat disadvantaged, because in interval $i = 1$, it competes using a core with a lower $X$ and possibly a higher $Y$ value. A designer could well be interested in the fact that one selection has a lower minimum operating power than the other, but this is orthogonal to SO and better evaluated with either the minimum or spread basic metrics.

### 6.8.4 Example: Comparing Selections

To demonstrate set overhead, the *Alternative* selection is compared to the *Baseline* selection (both shown in **figure 6.10**). A visual inspection suggests that the *Alternative* selection is slower, but for the designer to be able to perform an informed cost-benefit analysis, it is important to know how much better the *Baseline* selection is. **Figure 6.11** shows the set overhead of using the *Alternative* selection instead of the *Baseline* selection for the five example benchmarks. SO ranges from less than 5% up to 13%, with an average of 6%. I.e., on average, the average task will only be 6% slower on the *Alternative* selection than on the *Baseline* selection, but $bm_2$ will be 13% slower. If $bm_2$ represents a low-priority task and the *Alternative* selection is cheaper

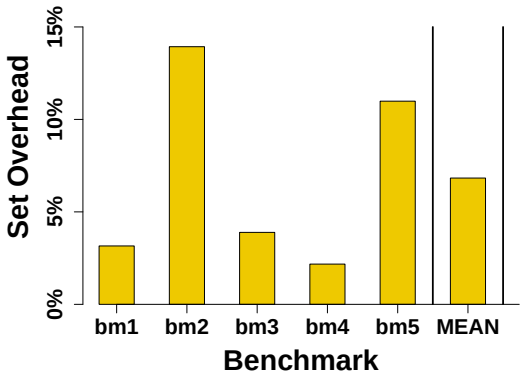


**Figure 6.11:** *Set overhead (time wasted) for $bm_{1\text{-}5}$ when using the* Alternative *selection instead of the* Baseline *selection (both shown in* ***figure 6.10****).*

to implement, then it may be possible to use the *Alternative* selection and save on engineering effort. If $bm_2$ represents an important task, then the *Alternative* selection of cores can be ruled out.

### 6.8.5 Example: Comparing DVFS and Heterogeneity

Set overhead can also be used to evaluate completely different technologies. This example demonstrates how a designer can use GO and SO together to choose whether to implement a heterogeneous processor or a homogeneous processor with DVFS. DVFS is a more mature technology than heterogeneity, but heterogeneity has the potential to be more effective.

The example assumes that a designer must choose between implementing a heterogeneous processor and implementing a homogeneous processor with DVFS. **Figure 6.12** shows the cores for these two alternatives for benchmark $bm_2$. The first alternative contains three heterogeneous cores from the candidate set (cores #319, #963, and #2778). The second alternative is the highest-power core, core #319, with DVFS applied. The DVFS data is extrapolated from the voltage and frequency data published by Lukefahr et al. [115]. Ten DVFS levels are assumed, as shown by the black dots along the dashed line.

The first step in the comparison is to measure gap overhead for the two alternatives. Since different technologies are used, the $\overline{Y_i}$ estimates are defined differently. For the heterogeneous case, the complete set of Pareto-optimal cores is used to determine $\overline{Y_i}$. For the DVFS case, $\overline{Y_i}$ is calculated from first principles. The heterogeneous selection has a gap overhead of 8%; the DVFS core's gap overhead is 2%. Gap overhead

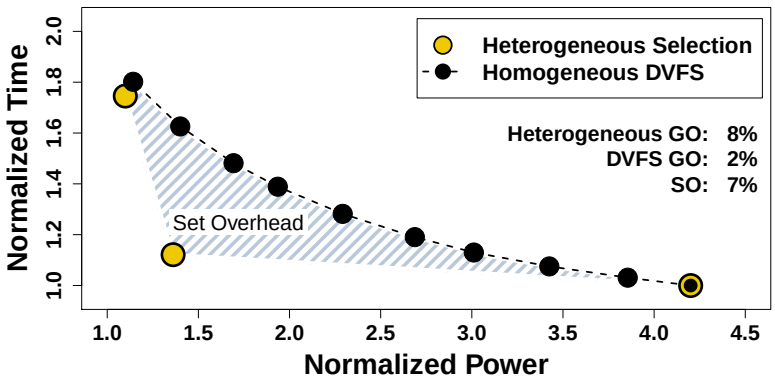


**Figure 6.12:** *Set overhead can be used to measure how much faster, on average, a heterogeneous set of cores is compared to a fast, DVFS-enabled core.*

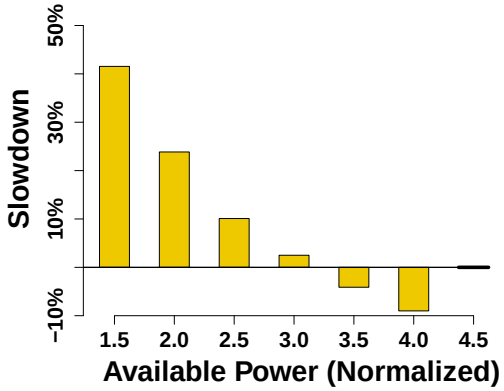


**Figure 6.13:** *The slowdown of the DVFS core over the heterogeneous selection varies depending on how much power is available. See* ***figure 6.12****.*

quantifies what is in this case obvious from visual inspection: that the ten DVFS levels closely approximate an infinite number of DVFS levels, while the three heterogeneous cores are not as close an approximation of the complete Pareto-optimal set of cores.

Gap overhead does not, however, help determine which alternative the designer should choose to implement. Gap overhead is lower for the DVFS core even though the heterogeneous set is visibly faster. Instead, comparing the two alternatives requires set overhead. In this case, $\overline{Y_i}$ in the SO equation is based on the complete set of cores. SO shows that on average, the DVFS core wastes 7% more time than the heterogeneous set. The shaded region in **figure 6.12** illustrates set overhead.

Set overhead measures the *average* slowdown across the entire range of power values. **Figure 6.13** shows the slowdown of the DVFS core over the heterogeneous selection at several discrete power levels. If there is enough power to run on the fastest core, then the heterogeneous selection and DVFS core are equally fast. If

available power (normalized) is 4.0, then slowdown is $-9\%$—the DVFS core is *faster* than the heterogeneous selection. However, as available power decreases, the relative performance of the heterogeneous selection improves considerably. When only very little power is available, DVFS can be 40% slower than the heterogeneous selection.

The example shows that GO and SO are complementary. Knowing the above results, the designer can reason about the costs and benefits of implementing either option. If the processor is intended for a power-limited setting, then the heterogeneous option is preferable. While this example compares heterogeneity and DVFS, a designer could also evaluate a heterogeneous processor with DVFS by suppling the metrics with a combinations of core types and DVFS levels.

## 6.9 Availability

*Availability* measures the likelihood that a selection of cores will be available to do work. I.e., it measures the probability that a given task can be run at all. The gap overhead metric only considers the available power PDF in the range where there is sufficient power to run the given benchmark on at least one core. GO is undefined at power levels that are below the power consumption of the lowest-power core, since the given task cannot be run at all. The availability metric quantifies this undefined range. The intuition behind the availability metric is described first, followed by its definition, a discussion, and an example of its use.

### 6.9.1 Intuition

Availability measures the likelihood that a set of cores can even run a task. For example, assume that the available power PDF ranges from the lowest-power benchmark-core combination to the highest-power benchmark-core combination. I.e., the scheduler will never make less power available to a task than is required to run the lowest-power task on the lowest-power core, and the scheduler will never make more power available to a task than is used by the most power-hungry task on the highest-power core. This guarantees that the lowest-power task can always run. However, due to behavior variations among tasks, other tasks will require more power, and will sometimes be unable to run. An example of this is shown in **figure 6.14** for the *Baseline* selection of cores. $bm_5$ requires more power to run on the lowest-power core than other benchmarks. For $bm_5$, the available power PDF falls in the normalized power range

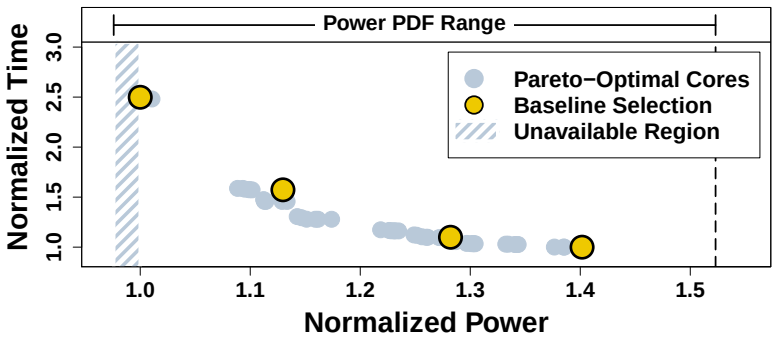


**Figure 6.14:** *At some power levels, $bm_5$ cannot be run at all on the* Baseline *selection, while other benchmarks can still be run. For $bm_5$, the power PDF falls in the normalized power range* $[0.98, 1.52]$.

$[0.98, 1.52]$, whereas for $bm_1$, the same power PDF falls in the normalized power range $[1.0, 1.56]$. Recall that a normalized power of 1.0 is the least amount of power required to run the benchmark on any core in the design space. The availability of the baseline selection is 100% for $bm_1$, but only 96% for $bm_5$.

### 6.9.2 Definition

Availability, Av, is measured on a per-benchmark basis. It is a function of the power PDF and the lowest-power core in the selection.

$$Av_{bm} = \frac{P_{max} - P_1}{P_{max} - P_{min}} \tag{6.4}$$

$Av_{bm}$ is the availability of the set of cores for benchmark $bm$.
$P_{min}$ is the minimum amount of power that could ever be allocated to a workload.
$P_{max}$ is the maximum amount of power that could ever be allocated to a workload.
$P_i$ is the power consumption of $bm$ on the $i^{th}$ core in the selection, ordered by increasing power.

The availability of a selection of cores for benchmark $bm$ is simply the fraction of the available power PDF that is greater than the power consumption of the lowest-power core for $bm$. If $Av_{bm} = 0.9$, then 10% of the time, there is insufficient power to run $bm$.

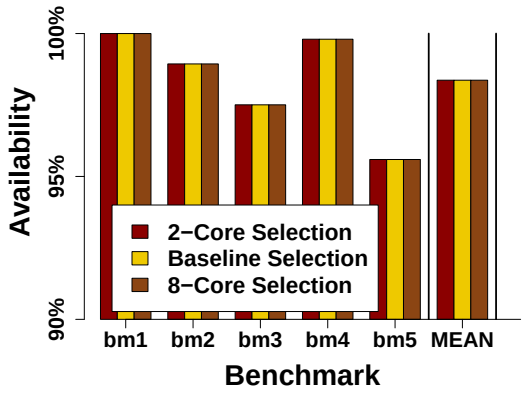


**Figure 6.15:** *The* 2-Core*,* Baseline*, and* 8-Core *selections all have the same availability, as all have the same lowest-power core.*

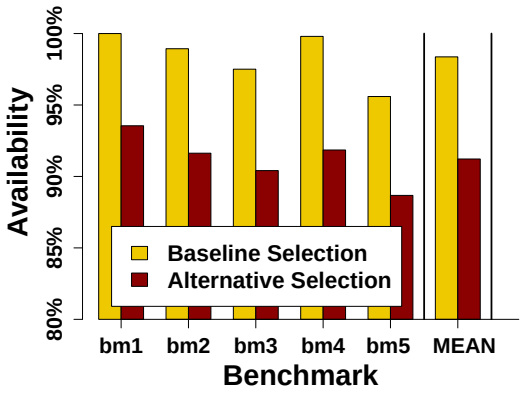


**Figure 6.16:** *The lowest-power core of the* Alternative *selection consumes more power than the lowest-power core on the* Baseline *selection, giving the* Alternative *selection a lower availability.*

### 6.9.3 Discussion

The availability metric is heavily influenced by the available power PDF, and is particularly sensitive to the lower-bound of the PDF. The intuition section above optimistically assumed for the sake of example that there will always be enough power to at least run the lowest-power task on the lowest-power core. On a real mobile processor, the probability that no new tasks can be run could be quite high. Availability would therefore be much lower. It is also possible that some tasks are so important that the scheduler will always make enough power available to them, potentially at the expense of other tasks. Availability for these tasks would be high, as the tasks would see a favorable power PDF.

### 6.9.4 Example: Comparing Availability

**Figure 6.15** shows the availability of the *2-Core*, *Baseline*, and *8-Core* selections for the example set of five benchmarks. Av is the same for all three selections, since the metric is a function of only the lowest-power core. As can be seen in **figure 6.2**, all three selections share the same lowest-power core.

**Figure 6.16** shows the availability of the *Baseline* and *Alternative* selections. The lowest-power core in the *Alternative* selection consumes more power than the lowest-power core in the *Baseline* selection (see **figure 6.1**). As a result, the *Baseline* selection has better overall availability.

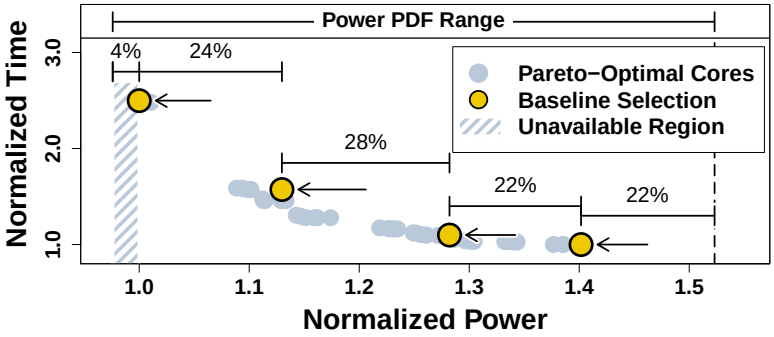


**Figure 6.17:** *Each core in the selection can be used a fraction of the time based on the power PDF, as indicated by the ranges and arrows. ES averages execution speed across all cores. The unavailable region lowers ES. Data is for* $bm_5$ *and the* Baseline *selection.*

## 6.10 Effective Speed

*Effective speed* measures the average throughput of a set of cores under an available power PDF. The metric is formulated similarly to the overhead metrics, but it is a throughput metric rather than a turnaround time metric. As such, it suffers from the same shortcomings as existing throughput metrics (see **section 5.6.1** (p. 63)). The intuition for the metric is given first, followed by its definition. A discussion expands on the interpretation and shortcomings of effective speed. The section concludes with an example.

### 6.10.1 Intuition

Existing metrics for measuring the throughput of a processor, such as STP (system throughput [52]), are independent of the amount of power that a processor consumes. These metrics can be used to measure maximum throughput or throughput under a fixed power budget. In contrast, the gap and set overhead metrics are turnaround time metrics that are based on a probabilistic model of the amount of power available at runtime. Sometimes it may be necessary to use a metric that is based on an available power distribution, but measures throughput. Effective speed is such a metric. Given a power PDF, each core in a selection can be used a certain percentage of the time. This is illustrated in **figure 6.17** with percentage ranges and arrows. At times, there is so little power available that the task cannot be run at all. Effective speed calculates the average speed over time for a selection of cores. Note that two of the cores in

**figure 6.17** appear behind the set of Pareto-optimal cores, since the *Baseline* selection is selected from the candidate set, but not all cores in the candidate set are Pareto-optimal for $bm_5$.

### 6.10.2 Definition

Effective speed, ES, is measured on a per-benchmark basis using only the cores from the selection that are Pareto-optimal for the given benchmark.

$$ES_{bm} = \left(\overline{T_{bm}} \times \frac{1}{Av_{bm}}\right)^{-1} \qquad \overline{T_{bm}} = \frac{T_N \times (P_{max} - P_N) + \sum_{i=1}^{N-1} T_i \times (P_{i+1} - P_i)}{T_{min} \times (P_{max} - P_1)} \quad (6.5)$$

$ES_{bm}$ is the effective speed of the set of cores for *bm*.

$Av_{bm}$ is the availability of the set of cores for benchmark *bm*.

$\overline{T_{bm}}$ is the weighted average scaled execution time of *bm* on the set of cores.

$P_{min}$ is the minimum amount of power that could ever be allocated to a task.

$P_{max}$ is the maximum amount of power that could ever be allocated to a task.

$T_{min}$ is the minimum possible execution time for *bm*.

$P_i$ is the power consumption of *bm* on the $i^{th}$ core in the set, ordered by increasing power.

$T_i$ is the execution time of *bm* on the $i^{th}$ core in the set, ordered by increasing power.

$N$ is the number of cores in the set.

ES uses absolute (not normalized) power and time values. There are no obvious applications of ES beyond evaluating performance with respect to power, so the metric is defined in terms of power and time rather than the more general $X$ and $Y$ used in the overhead metrics. The first component of the ES equation is $\overline{T_{bm}}$, which quantifies how long *bm* will take to run if there is enough power to run it on at least one core type. It is a normalized quantity—if $\overline{T_{bm}} = 2.0$, then *bm* will take twice as long to run on average, than it would on the fastest possible core. $\overline{T_{bm}}$ is the weighted average of each core's execution time. The weight for core $i$ is the likelihood that there is enough power to use core $i$, but not enough power to use core $i+1$. $\overline{T_{bm}}$ is itself weighted with the availability of the selection. For example, if $Av_{bm} = 0.5$, then *bm* will not run at all for half of the time, and the effective speed is cut in half.

Effective speed is the rate of execution expressed relative to the fastest possible core for the benchmark. I.e., it quantifies how fast the benchmark can run in reality,

relative to how fast it would run if it had access to unlimited power. For example, $ES_{bm} = 0.5$ means that on average, the set of cores will run *bm* at half the maximum possible speed.

### 6.10.3 Discussion

While effective speed is a throughput metric, it is not a replacement for the STP metric (see **section 3.5.2** (p. 33)). ES is still based on the runtime model that approximates the system with a power PDF, and it is evaluated for one task rather than for the entire system. I.e., in the nomenclature of Eyerman and Eeckhout [52], it is a *user-oriented* metric rather than a *system-oriented* metric.

The major limitation of effective speed is the difficulty of determining a good value of ES. For GO and SO, the best value possible is 0.0. A GO of 0.0 means that the selection of cores perfectly represents the set of all possible cores. An SO of 0.0 means that two selections have identical behavior on average. When GO or SO is greater than 0.0, it quantifies for the designer how much longer execution will take in the average case. In contrast, it is difficult to determine a good value of ES. The best possible value of ES is 1.0, which indicates that there is always enough power to run on the fastest core. However, if this is the case, then there is no need for heterogeneity. If ES is low, it could indicate that the processor can operate across a broad range of power values—a desirable feature—or that there are too few fast cores—an undesirable feature.

A second limitation of ES is that since it averages performance across cores, it hides performance variations. For example, the ES for a task alternating between running on a slow core and a fast core, and the ES of the task running on a medium core could be the same. However, a user is likely to be much more satisfied with the consistent performance of the latter. The overhead metrics assume that cores are exclusively allocated to tasks (which may be program phases) for the duration of the task. This guarantees consistent performance for the duration of the task, and provides a stronger foundation for user satisfaction studies.

The positive aspects of ES are its simplicity and flexibility. While ES is more difficult to interpret than other metrics, the concept of effective speed is intuitively obvious, and may therefore be more helpful for communicating results than other metrics.

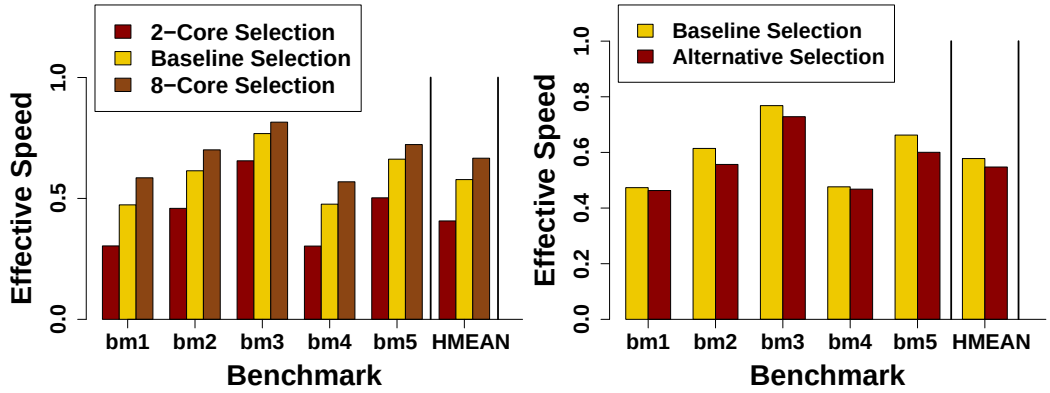


**Figure 6.18:** *Effective speed, ES, for the* 2-Core*,* Baseline*, and* 8-Core *selections (left), and the* Baseline *and* Alternative *selections (right). Larger ES values are better. ES correlates with GO (**figure 6.8**) and SO (**figure 6.11**)—more core types improve speed, and the baseline selection is faster than the alternative selection.*

### 6.10.4 Example: Comparing Throughput

**Figure 6.18** shows effective speed for the *2-Core*, *Baseline*, and *8-Core* selections, a comparison with increasing number of cores, and for the *Baseline* and *Alternative* selections, a comparison between two independent selections. This example assumes that the available power PDF is flat and ranges from the power consumption of the lowest-power benchmark-core combination to the power consumption of the highest-power benchmark-core combination, plus 10%. Since ES is a rate, it is summarized with the harmonic mean. Similarly to gap overhead, ES demonstrates that under tight power constraints, more core types enable faster execution (i.e., there is less time wasted). Moving from two to four to eight cores, ES increases from 0.40 to 0.58 to 0.66. Similarly to set overhead, ES demonstrates that the baseline selection is slightly faster than the alternative selection (ES $= 0.58$ and ES $= 0.55$, respectively).

## 6.11 Generality

*Generality* evaluates the extent to which core types are specialized to only some workloads, or are generally applicable to all workloads. The intuition behind the metric is described first, followed by its definition, a discussion, and a usage example.

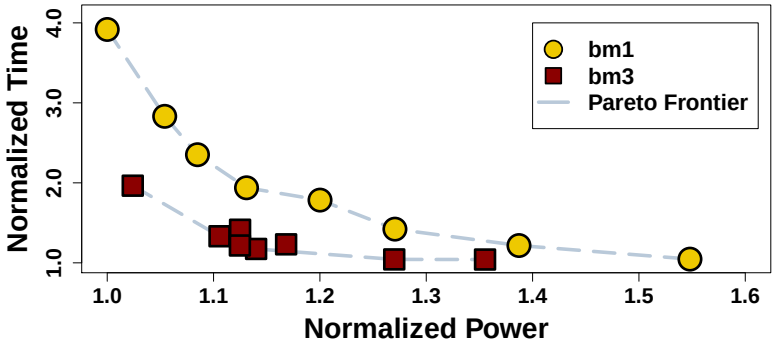


**Figure 6.19:** *Cores in the* 8-Core *selection, plotted for* $\mathrm{bm}_1$ *and* $\mathrm{bm}_3$. *All cores in the selection are Pareto-optimal for* $\mathrm{bm}_1$, *but only six are Pareto-optimal for* $\mathrm{bm}_3$.

### 6.11.1 Intuition

The uniformity, gap overhead, set overhead, availability, and effective speed metrics are all applied on a per-benchmark basis, and only consider those cores in the selection that are power-performance Pareto-optimal for the given benchmark. If a core in a selection is not Pareto-optimal for a given task, then it should never be used for that task, as there will be another core that is both faster and consumes less power. The variability in task behavior means that it is inevitable that not all cores in a selection can be Pareto-optimal for all types of tasks when the size of the selection is non-trivial (see **section 5.3.4** (p. 56)). An example of this is shown in **figure 6.19**. The cores in the *8-Core* selection are all Pareto-optimal for $bm_1$, as they are all on the Pareto frontier. Only six of the cores in the selection are Pareto-optimal for $bm_3$, since two of the cores are behind the Pareto frontier. This shows that $bm_3$ does not benefit from the eight cores as much as $bm_1$. (The selection also has poor spread and uniformity for $bm_3$, but these are orthogonal issues.) The generality metric measures the extent to which all types of tasks—all benchmarks in the representative suite—can take advantage of a selection of cores.

### 6.11.2 Definition

Generality is measured once for all benchmarks for a given selection of cores. It is represented by the Cyrillic Ђ (*dje*).

$$\mathbb{G} = \frac{1}{W} \sum_{i=1}^{W} \frac{|po(\mathcal{K}, i)|}{N} \tag{6.6}$$

| | |
|---|---|
| $\mathbb{G}$ | is the generality of the set of cores. |
| $W$ | is the number of workloads, or benchmarks. |
| $\mathcal{K}$ | is the selection of cores. |
| $po(\mathcal{S}, i)$ | returns the cores in set $\mathcal{S}$ that are power-performance Pareto-optimal for benchmark $i$. |
| $N$ | is the number of cores ($N = \lvert\mathcal{K}\rvert$). |

Generality is the fraction of cores in a selection that the average task can potentially use. It simply calculates the fraction of cores in a selection that are Pareto-optimal for each benchmark, and then takes the average. Generality falls in the range $[N^{-1}, 1.0]$. If the cores are generally applicable, then $\mathbb{G} = 1.0$. When $\mathbb{G} = N^{-1}$, then each core is specialized to only one benchmark—an unlikely, extreme outcome.

### 6.11.3 Discussion

Metrics like GO, SO, and ES encourage increasing the number of core types. At worst, they will show that adding a core has no effect, but in general, they will tend to show improvement with more core types. $\mathbb{G}$ counteracts the drive for more cores. As cores are added to a selection, the generality of the selection decreases, and each additional core benefits a smaller subset of benchmarks. The designer must decide whether the engineering effort required to add additional cores is justified when each additional core improves the processor by an ever smaller amount.

When generality is interpreted, it must be with consideration for the processor's target use case and the total number of core types, $N$. For example, if there are only four benchmarks in the representative suite used to design the processor ($W = 4$), if there are four core types ($N = 4$), and if $\mathbb{G} = 1.0$, then the set of cores is a good candidate for a mobile processor. Each of the four cores is usable by all benchmarks. Which core is used at runtime is determined by runtime requirements, such as the amount of available power. If, on the other hand, $\mathbb{G} = 0.25$, then each core is usable by only one benchmark—each core's microarchitecture is specialized to the specific behavior of each benchmark. Which core is used at runtime is determined by the task, without regard for runtime power constraints. Such a set of cores does not offer the runtime flexibility required by a mobile processor, though it could be appropriate for

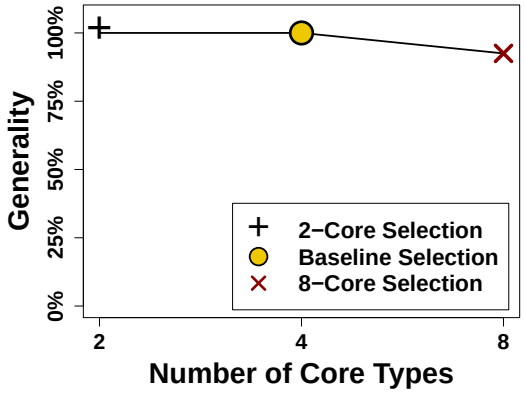


**Figure 6.20:** *As the number of core types increases, their generality—the number of cores each task can use—decreases. Data is shown for the* 2-Core*,* Baseline*, and* 8-Core *selections and benchmarks* $\text{bm}_{1\text{-}5}$*.*

a processor that can be guaranteed to always have a large amount of power available (e.g., some server processors). Finally, if $W = 4$, $N = 16$, and $\text{Ъ} = 0.25$, then the set of cores could be appropriate for a mobile device, as each benchmark has, on average, four core types it can use. The core that is used is again dependent on runtime requirements. There is still substantial specialization to different tasks, but the number of cores ensures flexibility. Whether such a processor is physically or financially feasible is a separate issue. This example demonstrates that Ъ must not necessarily be minimized or maximized; a good value of Ъ is determined by other design requirements.

Generality also has an effect on scheduling. If a set of cores has low generality, then a more intelligent scheduler is required to determine which cores any given task can be run on.

It should be noted that when existing literature discusses *monotonicity* (see **section 3.2.4.1** (p. 22) and **section 5.3.2** (p. 53)), often the concept under consideration is, in fact, generality. Monotonicity is subtly different from generality, but it can often be used as a proxy for generality. The next metric measures monotonicity.

### 6.11.4 Example: Generality of Selections

An example of generality is shown in **figure 6.20** for the *2-Core*, *Baseline*, and *8-Core* selections and the five example benchmarks, $bm_{1\text{-}5}$. The *2-Core* and *Baseline* selections are applicable to all benchmarks, but a few of the cores in the *8-Core* selection should not be used by some benchmarks. Even for the *8-Core* selection,

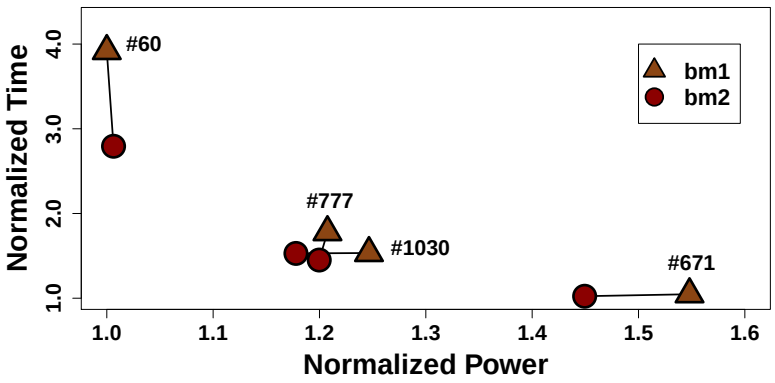


**Figure 6.21:** *Four cores are plotted by power and time for $bm_1$ (triangles) and also for $bm_2$ (circles). The ordering of cores changes depending on the benchmark, as identified by the black lines. Power and time are normalized to each benchmark's best value.*

$\mathbb{T}$ is quite high. This is because of the small number of example benchmarks and because the selections of cores have been carefully selected by hand.

## 6.12 Monotonicity

*Monotonicity* evaluates how dependent the ordering of cores is on the type of task being executed. It is closely related to the generality metric. Monotonicity has implications for scheduling, and as described in **section 5.3.2** (p. 53), affects whether a selection of cores is applicable to a mobile processor. The intuition behind the metric is described first, followed by its definition, a discussion, and a usage example.

### 6.12.1 Intuition

**Figure 6.21** shows four cores plotted for $bm_1$, and the same four cores plotted for $bm_2$. These cores are neither the *Baseline* nor *Alternative* selections, but have been chosen specifically for this example. It can be seen that the order of cores is dependent on the benchmark. Core #1030 consumes more power than core #777 for $bm_1$, but the roles are reversed for $bm_2$. For this example, all cores are Pareto-optimal for both benchmarks—for the four cores and two benchmarks, $\mathbb{T} = 1.0$. The monotonicity metric measures the extent to which the order of cores is benchmark-dependent.

### 6.12.2 Definition

Monotonicity is measured once for all benchmarks for a given selection of cores. It is derived from Spearman's rank correlation coefficient (Spearman's ρ; see e.g., Conover [36, p. 314]), and is represented with the Hebrew ר (*resh*). The implementation of ר in this thesis is based on the R implementation of ρ [145].

$$ר = \frac{1}{W(W-1)/2} \sum_{i=1}^{W-1} \sum_{j=i+1}^{W} \rho\Big(o(X_i), o(X_j)\Big) \tag{6.7}$$

| | |
|---|---|
| ר | is monotonicity. |
| $W$ | is the number of workloads, or benchmarks. |
| $X_i$ | is the set of constrained metric values (power values in this case) when the core types execute workload $i$. |
| $o(\mathcal{S})$ | returns the permutation that will sort the elements in set $\mathcal{S}$ into ascending order. |
| $\rho(P_1, P_2)$ | measures Spearman's rank correlation coefficient between permutations $P_1$ and $P_2$. |

For each benchmark, core types are ordered by increasing power, and ρ is calculated for all pairwise combinations of benchmarks. ρ compares the order of items in two sets. It ranges from 1.0 when the order is the same, to $-1.0$ when the order is reversed. For ר, ρ values are summed and divided by the number of comparisons. When $ר = 1.0$, the processor is monotonic. As the order of cores becomes more benchmark-dependent, ר decreases.

### 6.12.3 Discussion

When existing literature discusses *monotonicity*, it is often referring to *generality* (see **section 3.2.4.1** (p. 22) and **section 5.3.2** (p. 53)). The two concepts are closely related but subtly different. A generality of less than 1.0 is often associated with a monotonicity less than 1.0, because if one benchmark's interaction with cores' microarchitectures causes the order of cores to change compared to the order for another benchmark, then it is likely that the benchmark's behavior also causes some cores to fall behind the power-performance Pareto frontier. However, as demonstrated by **figure 6.21**, this is not necessarily the case. Monotonicity is a reasonable proxy for generality, but the two do not correlate exactly.

Monotonicity has a similarly dampening effect on the drive to add core types as generality. Whereas a decrease in generality shows a designer that some cores are no longer useful to some types of tasks, a decrease in monotonicity highlights an increased scheduling complexity. As ר decreases, it becomes increasingly unclear which cores consume less power and which cores consume more. It therefore becomes increasingly difficult for a scheduler to schedule a task to a core that consumes an acceptable amount of power based on runtime requirements.

Sets of cores that have been dubbed *monotonic* in existing literature can more precisely be defined as sets where $\beth = 1.0$ and $ר = 1.0$. I.e., all cores can be used by all types of tasks, and the order of cores is always the same. The use of *non-monotonic* in existing literature is less clear. It tends to describe sets of cores where generality is minimal, regardless of monotonicity. I.e., if each type of task can only be run on one type of core, then the power order of the cores is irrelevant.

Availability, generality, and monotonicity used together can be a powerful combination. Availability ensures that the selection of cores is usable in a sufficiently broad set of circumstances. Generality ensures that a sufficiently large number of tasks can use each core. Monotonicity ensures that scheduling is sufficiently simple. Once these requirements are met, localized non-uniformity can be used to ensure coverage of the candidate set, gap overhead can be used to determine how many total cores there should be, and set overhead can be used to show that the selection is better than alternative selections (e.g., from competitors).

### 6.12.4 Example: Workload Divergence

The monotonicity metric, ר, compares the ordering of cores. If ר is less than 1.0, it indicates to the designer that benchmark behavior has diverged, and different benchmarks are responding differently to the various core types. As an example, for the *2-Core* and *4-Core* (*Baseline*) selections from **figure 6.2**, $ר = 1.0$, but for the *8-Core* selection, $ר = 0.76$. This shows that it is difficult to select eight cores that all have a consistent power order—with eight cores, it is almost inevitable that some benchmarks will cause changes in the core orders.

Monotonicity can be used to gauge how difficult a set of cores is to schedule for. If ר is 1.0, then the runtime scheduler will always know the order of cores from low power to high performance, because the order is the same for all workload types. As ר decreases, it will be increasingly difficult for a scheduler to determine where

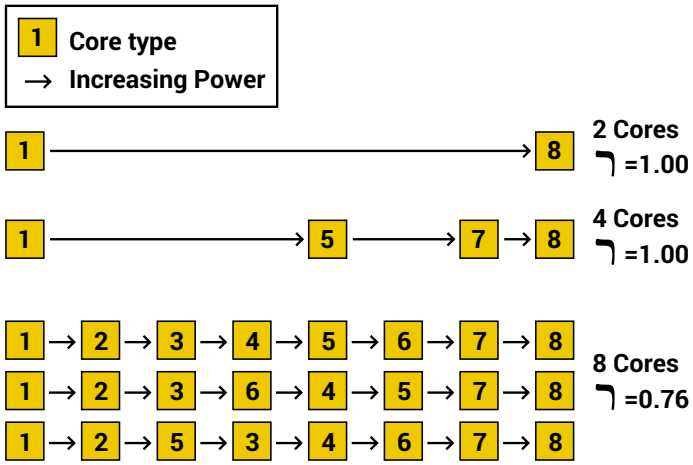


**Figure 6.22:** *The* 2-Core *and* Baseline *(*4-Core*) selections can be ordered by increasing power. For eight cores, there are three possible orderings, depending on the benchmark.* ¬ *evaluates the divergence in the orderings. See* ***figure 6.2*** *for cores. For brevity, cores are labeled 1-8.*

to schedule tasks. **Figure 6.22** illustrates this for the *2-*, *4-*, and *8-Core* sets from **figure 6.2**. For brevity, the cores are simply labeled 1-8. Core numbers are given in **section 6.4**. In the *2-* and *4-Core* cases, there is only one way to order cores from low power to high performance, regardless of which benchmark is considered. In the *8-Core* case, there are three possible orderings, depending on which benchmark is used. If, for example, a task running on the $6^{th}$ core must be moved to a less power-hungry core, then it is not immediately obvious whether the $5^{th}$ core should be considered. For some benchmarks, the $5^{th}$ core consumes less power than the $6^{th}$ core, but for others, it consumes more. This does not mean that a processor where $\neg < 1.0$ should never be designed, but it indicates to the designer that more effort will be required from the operating system developers to take advantage of the processor.

## 6.13 Power PDF Considerations

The runtime model in **section 5.4** (p. 58) described a probabilistically varying available power distribution, which forms the basis for the metrics presented in this chapter. Given that different types of tasks have different priorities and requirements, it is likely that in a final product, the distribution of power values that a scheduler makes available to a given task will be dependent on the task. For simplicity and consistency,

the metric definitions in this chapter have assumed a flat power distribution. This section considers how a known, non-flat distribution can be applied to the metrics. The distribution could be derived empirically or from first principles.

The shape of the power PDF does not affect the generality and monotonicity metrics, or the KS test. Generality evaluates the number of cores that are Pareto-optimal for a given benchmark, and Pareto optimality is independent of the probability density of power values. Monotonicity is based on ordinal numbers through Spearman's $\rho$, and is similarly unaffected by the PDF. The KS test compares the distribution of the selection to the distribution of the candidate set. A PDF applied to the selection would also be applied to the candidate set, and the two would cancel out.

The application of a non-flat PDF to gap overhead, set overhead, availability, and effective speed is trivial. For the two overhead metrics, the $\alpha$ value (**equation 6.2** and **equation 6.3**) controls the weight of each interval. Applying the PDF is simply a case of setting $\alpha$ to the integral of the PDF in the relevant range, keeping in mind that the PDF must be normalized such that it is zero in the undefined region below the lowest-power core. For effective speed, the PDF similarly adjusts the weight of each core (the power subtraction term in the summation of **equation 6.5**). In the case of availability, availability is the integral of the PDF greater than the power of the lowest-power core. For these four metrics, applications of the PDF are benchmark-specific.

The application of a PDF to the localized non-uniformity metric is more involved, because the method of application is dependent on the precise feature the designer wishes to measure. For localized non-uniformity, the PDF is also applied on a per-benchmark basis. The definition in **section 6.6.3.1** assumed that the cores would be flattened to one dimension using Euclidean distance. This is a natural default option, since $\beth$ then measures how good the coverage of the candidate set is. However, a designer may, instead, choose to apply $\beth$ to just the time coordinates of cores. This would help the designer evaluate how gracefully performance can change—large gaps between time values lead to large changes in performance that are jarring to the user. Similarly, the designer could choose to apply $\beth$ to just the power coordinates. When cores provide a more uniform selection of power values, then it is easier for a scheduler to allocate available power. Since cores follow a trade-off curve, more uniformity along one axis inevitably leads to less uniformity along the other. The flattened Euclidean approach is a middle ground. If the designer chooses to apply $\beth$ to just the power axis, he could further choose to apply the power PDF to the axis. In this case, a low $\beth$ value would mean that under the power PDF, each core sees a similar level of

utilization. Finally, the designer could apply the power PDF to the power axis, and also use Euclidean flattening. ב would then evaluate a combination of core utilization rate and the gracefulness of performance changes. If ב is evaluated using only the time axis, then the application of a power PDF has no effect.

## 6.14 Summary

To evaluate heterogeneous processors, it must be possible to quantitatively evaluate the selection of CPU core types implemented on the processor. Traditional techniques can evaluate individual cores, but new metrics are required to evaluate sets of cores as they operate under the volatile power budgets of heterogeneous processors. This chapter began with a set of goals and a small number of assumptions based on which it is possible to define such metrics. After clarifying some basic evaluation methods, metrics were defined for quantifying: the diversity of sets of cores, the performance of sets of cores relative to an ideal limit, the performance of sets relative to other sets, the availability of sets, the effective throughput of sets, and the general applicability and monotonicity of sets. Each metric was introduced with an intuitive description and demonstrated using example simulation data. Without quantitative metrics, the designer of a heterogeneous processor is effectively blind, relying solely on intuition to select cores. This collection of metrics forms a set of tools that allows the designer to quantify intuition and approach the heterogeneous core selection problem with mathematical rigor.

# Chapter 7:

# The LUCIE Algorithm for Core Selection

Converting a candidate set of potential CPU cores into a heterogeneous processor requires a solution to the selection problem. Most cores in the candidate set must be discarded; only a small number can be selected for implementation. This chapter describes the *LUCIE* algorithm, a selection strategy that iteratively removes cores from the candidate set while preserving diversity and thereby enabling flexibility for a set of representative benchmarks. There are three variations to LUCIE and two mechanisms for providing LUCIE with prior knowledge from an expert.

## 7.1 Introduction

The demand for heterogeneity in the mobile space is driven by the need for runtime power flexibility. Mobile devices cannot consume large amounts of power, yet users still desire high performance. A heterogeneous processor implements a diverse selection of cores, and has the potential to provide the best performance to a given task within the amount of power that is available at a given time.

To design a heterogeneous processor, the designer must solve the *selection problem*—the designer must choose which different types of CPU cores should be implemented on the processor. As described in **section 5.3.1** (p. 52), the selection problem is distinct from design space exploration (DSE). DSE is used to find a potentially large set of cores that *could* be implemented. Core selection is used to choose which cores from DSE *should* be implemented. Some existing selection methods were summarized in **section 5.7** (p. 69).

The selection problem is particularly difficult for mobile devices. A mobile processor must implement cores ranging from low-power to high-performance so that it can be fast when possible, and operate at low power when required. It is not sufficient to select cores that only maximize the performance of different types of programs, as done by the GA selection strategy (**section 5.7.3** (p. 70)); mobile devices also require low-power cores. Neither is it sufficient to select cores to maximize consistency, as done by the Clustering strategy (**section 5.7.4** (p. 71)); a selection technique that maximizes consistency necessarily precludes taking advantage of program diversity.

The *LUCIE* selection algorithm addresses the problem of selecting cores for mobile devices. The algorithm is presented in the current chapter and evaluated in **chapter 8**. The remainder of this chapter is arranged as follows: **Section 7.2** gives a high-level overview of the LUCIE algorithm. **Section 7.3** describes the significant characteristics of the algorithm. A selection algorithm could be formulated in any number of different ways; this section justifies the specific formulation of LUCIE. **Section 7.4** defines the coordinate space in which LUCIE operates. The following three sections present three variants of the LUCIE algorithm in order of increasing sophistication. These are *basic LUCIE*, *biased LUCIE*, and *weighted-biased LUCIE*, or E-LUCIE, BE-LUCIE, and WB-LUCIE for short. Each variant builds on the previous one. The subsequent section describes *core pinning*. The processor designer can use core pinning to force LUCIE to select specific cores. Finally, the possibility of using a power PDF (probability density function) is introduced. A PDF directs LUCIE to select cores to conform to a known available power distribution (see the runtime model in **section 5.4** (p. 58)). As LUCIE is a selection algorithm, this chapter does not consider DSE, but uses the candidate set described in **section 4.3.2** (p. 42).

## 7.2 LUCIE Overview

**Section 5.2** (p. 50) identified runtime flexibility as the primary motivation and justification for heterogeneity. **Section 5.5** (p. 60) identified spread and uniformity as the two qualitative features a selection of heterogeneous cores must possess to enable runtime flexibility. **Section 5.3.2** (p. 53) defined the selection problem as a problem of maintaining as much of the diversity of the candidate set as possible with the number of cores that the processor designer is willing to implement. Based on this problem description, the LUCIE algorithm has two related goals: to select cores at a variety of power-performance points ranging from low power to high performance,

and to ensure that different types of programs (different benchmarks) have access to a range of power-performance points. Since the power and performance of a core are dependent on the characteristics of the program being run, it is crucial that a selection algorithm considers how each of the representative benchmarks performs on each core. A core that is fast and power-hungry for one benchmark might be slow, yet power-hungry for another. LUCIE also attempts to avoid disproportionately high-power and disproportionately slow cores for reasons described in the spread discussion, **section 5.5.1** (p. 61).

LUCIE stands for *L*east *U*seful *C*onfiguration *I*terative *E*limination. As described in **section 5.3.4** (p. 56), program behavior can have a significant effect on a core's power and execution speed. The LUCIE algorithm is based on the insight that maximizing performance on power-constrained, mobile devices requires taking advantage of the task-dependent performance variations of cores. (This is the opposite of the approach adopted by the Clustering selection strategy [71], which selects cores for server processors and prioritizes consistency in each cores' behavior.) The underlying contribution of the LUCIE algorithm is recognizing that one core can fulfill different roles for different types of benchmarks. A core might provide relatively high performance at a high power cost to one benchmark, it might provide moderate performance at a moderate cost to another benchmark, and it might not even be Pareto-optimal for a third benchmark. LUCIE selects cores based on the significance of their contribution to each benchmark, as compared to the significance of other cores.

The LUCIE algorithm treats the complete candidate set of cores as the gold standard for diversity. The candidate set comes from a design space search (see **figure 5.2** (p. 53)). LUCIE considers all relevant benchmarks, finds the core that contributes least to diversity, removes it from the set, and repeats until the desired number of cores remains. I.e., it is a subtractive rather than an additive selection strategy. The metrics defined in **chapter 6** can be used to evaluate various features of sets of cores selected by LUCIE or by other algorithms. However, for reasons described in the next section, LUCIE only optimizes these metrics indirectly.

## 7.3 Algorithmic Considerations

The significant feature of LUCIE is that it is not a multiobjective optimization algorithm (MOA) in the traditional sense. While the selection problem resembles a multiobjective optimization problem, there are key differences between the two. An MOA

attempts to efficiently traverse a solution space to find non-dominated solutions—points that are Pareto-optimal for the objectives that must be optimized. MOAs are based on the observation that there might not be an objective function that can combine all the multiple objectives into a single figure of merit, so the algorithms must return a population of non-dominated solutions rather than a single solution. Since CPU cores attempt to maximize performance while minimizing power, MOAs are a good option for design space exploration. The selection problem is, however, a problem of selecting a complementary set of non-dominated solutions (cores in the candidate set) for a processor. An MOA cannot differentiated among the cores in the candidate set, because by definition, they are all non-dominated and equally "good."

A naïve approach would be to turn the selection problem into a separate multiobjective optimization problem that attempts to optimize the selection of cores using, for example, the metrics in the previous chapter. While this is possible, it is unwise and unhelpful to the processor designer. **Chapter 6** presented eight metrics. Excluding the KS test, which is superseded by כ, and SO, which is a comparison to another selection, there are six metrics that could be optimized. However, כ, GO, Av, and ES operate on a per-benchmark basis. The number of objectives is therefore multiplied by the number of benchmarks. For even the trivial set of five benchmarks in **chapter 6**, there would be 22 objectives ($4 \times 5 + 2$). For the full set of benchmarks from **section 4.3.1** (p. 41), the number objectives is 134. The developer of an MOA may believe that he is doing a good deed by finding the don-dominated solution set in an objective space with 134 dimensions. However, by doing so, he assumes that the processor designer can choose one of the solutions from the 134-dimensional objective space. The designer's choice would be dependent on a long series of impossible to conceptualize trade-offs, such as determining whether trading a 10% increase in כ for benchmark 1 ($bm_1$) is counterbalanced by a 3% decrease in GO for $bm_4$. Applying an MOA to the selection problem transforms the problem of selecting some power-performance Pareto-optimal cores into a problem of optimizing potentially hundreds of objectives, a farcical increase in complexity.

There is the further issue that the metrics in **chapter 6** are designed to offer insight and enable detailed analysis of a selection of cores—they are intended to increase the amount of information known about a selection. Using the metrics as black-box quantities turns the insight into uninformative noise.

The LUCIE algorithm avoids the difficulties of an MOA by using a subtractive selection methodology. An additive selection methodology would require an algorithm

that repeatedly selects and evaluates cores. This would be combinatorially complex and would require an MOA or similar approach. In addition to the problems noted above, an MOA would use randomness in the selection, leading to non-determinism. The non-determinism can, in turn, lead to an endless cycle of rerunning the algorithm with tweaked parameters in the hope of finding a better solution. Rather than select which cores in the candidate set should be used, LUCIE iteratively selects which cores should not be used. The cores that remain comprise the selection. A subtractive approach allows LUCIE to be deterministic while avoiding the complexities of multiobjective trade-offs.

## 7.4 Selection Space Normalization

LUCIE performs core selection in a normalized, unit-less metric space that is identical to the one used by the ↄ metric (**section 6.6.3.1**). Power and execution time are expressed as multiples of lowest power and shortest execution time to account for variations among benchmarks. For each core-benchmark combination, power and execution time are divided by the best power and the best execution time for that benchmark by any core in the design space, as shown in **equation 7.1**. Normalization is carried out once on a dataset before the LUCIE algorithm is run.

$$P_{c,b} = \frac{\mathrm{P_{raw}}(c,b)}{\min\left(\mathrm{P_{raw}}(\mathcal{C},b)\right)} \qquad T_{c,b} = \frac{\mathrm{T_{raw}}(c,b)}{\min\left(\mathrm{T_{raw}}(\mathcal{C},b)\right)} \tag{7.1}$$

| | |
|---|---|
| $P_{c,b}$ | is the normalized (unit-less) power of core $c$ executing benchmark $b$. |
| $T_{c,b}$ | is the normalized (unit-less) execution time of core $c$ running benchmark $b$. |
| $\mathrm{P_{raw}}(c,b)$ | is the power consumption of benchmark $b$ on core $c$ in Watts, as reported by a power model. |
| $\mathrm{T_{raw}}(c,b)$ | is the execution time of benchmark $b$ on core $c$ in seconds, as reported by a simulator. |
| $\mathcal{C}$ | is the complete set of cores. |
| $\min(\mathcal{S})$ | returns the minimum of set $\mathcal{S}$. |

Power and time are normalized to enable fair comparisons between cores. As an example, knowing that benchmark $bm_A$ draws 500mW on a core and that benchmark $bm_B$ draws 1W on the same core is not particularly informative. Knowing that $bm_A$

**Algorithm 1** E-LUCIE (basic LUCIE) using default $\underline{\mathbf{C}}$ and $\underline{\mathbf{E}}$

```
while |C| > N do
    for all c in C do
        C(c)                                   ▷ Equations 7.2, 7.3, 7.4
    end for
    c ← FINDMINCOSTCORE(C)
    remove c from C
    for all b in B_c do
        cm ← arg min_k E(c, k, b)              ▷ Equation 7.4
        append b to list for cm if not present
    end for
end while
```

runs at 2.0× its minimum power and $bm_B$ runs at 1.1× its minimum power on the core suggests that the core has good power characteristics for $bm_B$ but not for $bm_A$, despite the fact that the absolute power for $bm_B$ is greater than for $bm_A$. The remainder of this chapter uses normalized power and time instead of absolute values. *Average normalized power* and *average normalized time* summarize the normalized power and time of a given core across all benchmarks using the arithmetic mean. Average normalized time is therefore equivalent to the ANTT metric [52]. Normalization aids in interpreting power and time values and provides a level of robustness against unreliable tools (see **section 5.3.5** (p. 58) and **section 6.2** (p. 75)).

## 7.5 Basic LUCIE

This section defines the baseline version of the LUCIE algorithm, *E-LUCIE*. Later sections add improvements to the basic algorithm. The definition is followed by an example.

### 7.5.1 Definition

E-LUCIE is shown in **algorithm 1**. The algorithm begins with the set of cores, $\mathcal{C}$, where every core in $\mathcal{C}$ is power-performance Pareto-optimal for at least one benchmark. I.e., $\mathcal{C}$ is the *candidate set*. The candidate set comes from a design space exploration (DSE), see **figure 5.2** (p. 53). An oracle candidate set is assumed here,

as described in **section 4.3.2** (p. 42). $N$ is the number of different cores LUCIE should select. It is set by the designer. Each core in $\mathcal{C}$ has a list of one or more benchmarks for which it is Pareto-optimal. The length of this list is the core's *affinity*—if a core is optimal for many benchmarks, then it has a high affinity, and if it is optimal for a few, then it has a low affinity. The affinity of a core is distinct from the generality metric, $\mathfrak{G}$, which is only calculated for the final selection of cores. LUCIE first iterates over all cores in $\mathcal{C}$, and calculates each core's *cost* (defined below). It then finds the core with the minimum cost, core $c$, and removes it from set $\mathcal{C}$. Finally, it iterates over all benchmarks that were associated with core $c$—the set $\mathcal{B}_c$. For each benchmark, $b$, in $\mathcal{B}_c$, it finds a destination core, $cm$, and adds $b$ to the benchmark list for core $cm$. $cm$ is the core nearest to $c$ for benchmark $b$ based on a distance metric, $\underline{\mathbf{E}}$. If $b$ was already associated with $cm$, then the benchmark list for $cm$ is unchanged. Otherwise, $b$ is added to the list for $cm$, and the affinity of $cm$ increases. The process repeats until the desired number of cores remain.

The cost of core $c$, $\underline{\mathbf{C}}(c)$, is given in **equation 7.2**. For each benchmark, $b$, associated with core $c$, the cost, $\underline{\mathbf{D}}$, of displacing the benchmark from $c$ is calculated. The total cost of core $c$ is the sum of the individual displacement costs. A core can have a high cost if there are many benchmarks associated with it, or if a few benchmarks have a large displacement cost (i.e., if the core is very important to a few benchmarks).

$$\underline{\mathbf{C}}(c) = \sum_{b}^{\mathcal{B}_c} \underline{\mathbf{D}}(c,b) \tag{7.2}$$

**Equation 7.3** defines the displacement cost, $\underline{\mathbf{D}}(c,b)$. This is a measure of how useful core $c$ is to benchmark $b$. If $\underline{\mathbf{D}}$ is large, then core $c$ occupies a unique position in the design space for the benchmark. If it is small, then there exists another core that has a similar power-performance trade-off for $b$. If core $c$ is to be removed, $b$ should be displaced to the nearest core. The distance metric, $\underline{\mathbf{E}}$, is used to find the distance between two cores for benchmark $b$. The closest core to core $c$ for benchmark $b$ is defined as core $cm$. The displacement cost is the cost of moving benchmark $b$ from $c$ to $cm$.

$$\begin{aligned} \underline{\mathbf{D}}(c,b) &= \underline{\mathbf{E}}(c,cm,b) \\ cm &= \arg\min_{k} \underline{\mathbf{E}}(c,k,b) \end{aligned} \tag{7.3}$$

Finally, **equation 7.4** defines the distance, $\underline{\mathbf{E}}$, between core $c_1$ and core $c_2$ for benchmark $b$. $\underline{\mathbf{E}}$ is simply Euclidean distance in normalized power-performance space.

Since no two benchmarks have exactly the same power-performance behavior, each benchmark has a different $\underline{\mathbf{E}}$ for the same pair of cores. $\underline{\mathbf{E}}$ uses the quantities ΔP and ΔT in **equation 7.5**. ΔP is the difference between core $c_1$ and core $c_2$ for benchmark $b$ along the normalized power axis; ΔT is the difference along the normalized time axis. A positive ΔP or ΔT means that moving from $c_1$ to $c_2$ is an improvement for $b$. Since the cores in $\mathcal{C}$ follow a Pareto frontier, a positive ΔP will generally be paired with a negative ΔT, and vice-versa.

$$\underline{\mathbf{E}}(c_1, c_2, b) = \sqrt{\Delta\mathrm{P}(c_1, c_2, b)^2 + \Delta\mathrm{T}(c_1, c_2, b)^2} \tag{7.4}$$

$$\begin{aligned} \Delta\mathrm{P}(c_1, c_2, b) &= P_{\mathrm{c_1,b}} - P_{\mathrm{c_2,b}} \\ \Delta\mathrm{T}(c_1, c_2, b) &= T_{\mathrm{c_1,b}} - T_{\mathrm{c_2,b}} \end{aligned} \tag{7.5}$$

### 7.5.2 Example

**Figure 7.1** shows the progression of the LUCIE algorithm as it eliminates cores. This example uses results from the 12 SPECint 2006 benchmarks (see **section 4.3.1** (p. 41)). The candidate set in this case contains 266 different cores. Cores are plotted by average normalized power and time. Since averages hide per-benchmark variations, gray boxes in the final plot show the full range of power and time values for each of the four cores. The right edge of each gray box is the highest (normalized) power consumption of any benchmark on the given core, etc. This illustrates that a core's behavior is dependent on the type of program being run.

The candidate set contains two clusters of cores that clearly fall onto the power-performance Pareto frontier, and an additional cluster of high-power cores. The high-power cores achieve a short execution time (are Pareto-optimal) for only a few benchmarks, and their affinity is therefore low. Since **figure 7.1** is averaged across all benchmarks, the high-power cores appear above the Pareto frontier. These fast cores are undesirable for two reason: First, there are cores with similar speed characteristics but much lower power consumption. Second, a heterogeneous processor can only implement a limited number of core types, and in general, it is therefore better for cores to be broadly applicable. **Equation 7.2** ensures that the low-affinity cores have a low cost and are removed.

**Figure 7.1** (bottom-right) shows that LUCIE thins the large candidate set down to four, roughly evenly spaced cores that approximate the original Pareto frontier. If

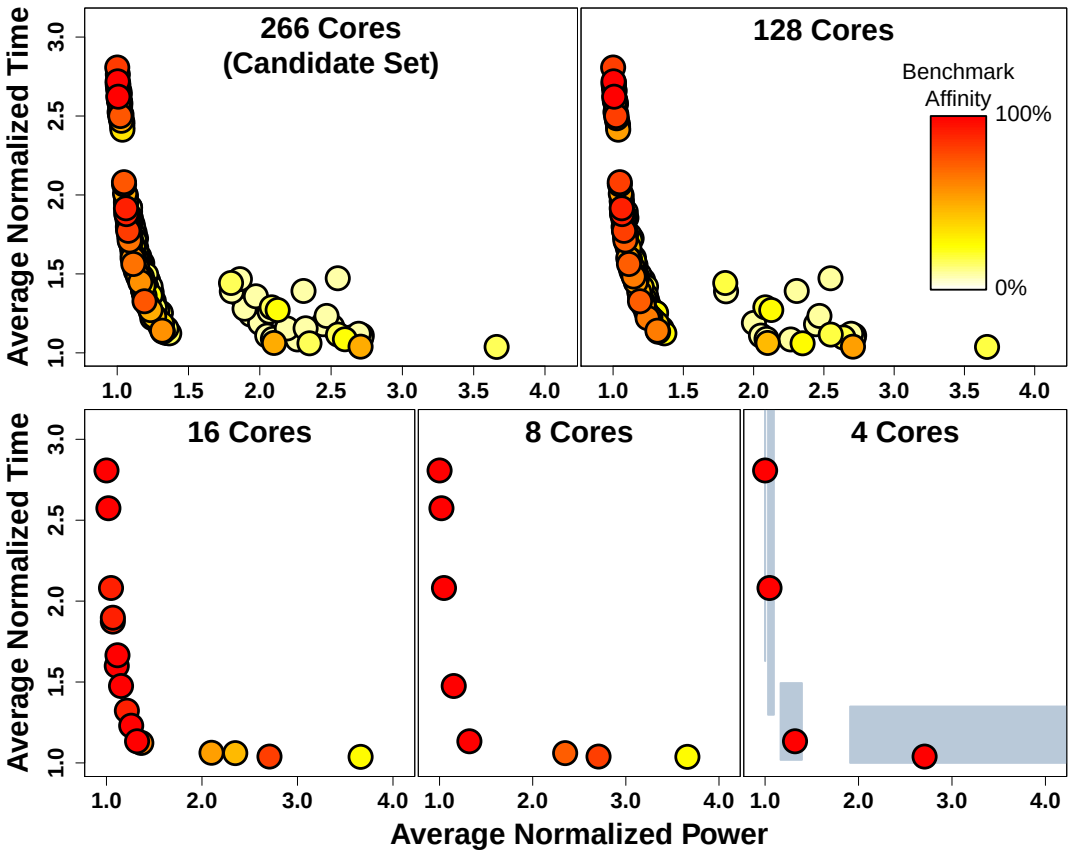


**Figure 7.1:** *LUCIE approximates the candidate set with an ever decreasing number of cores. Gray boxes show the full power and time range of the final four cores. Results are for E-LUCIE and* ***SPECint 2006****.*

LUCIE is allowed to select eight or sixteen cores, then the Pareto frontier is covered more densely, and some low-affinity cores are included.

LUCIE approximates the full, Pareto-optimal set with a small number of cores, while balancing the competing demands of design space coverage and per-benchmark specialization. When four cores are selected, the cores are effectively general-purpose—they all have a high affinity. When the number of cores increases to eight and beyond, some low-affinity cores appear. These provide significant benefits to only a few benchmarks. LUCIE does not have a stopping criterion, but can be run until only one core remains. It is up to the designer to decide how many core types should be implemented, as this will depend on the specific design space, the target benchmarks, available engineering effort, etc.

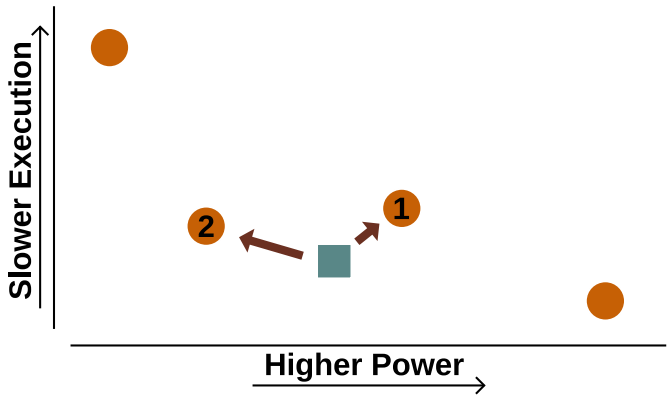


**Figure 7.2:** *E-LUCIE only considers distance when displacing benchmarks. If the square core were to be removed, benchmarks would be displaced to core 1. BE-LUCIE also considers improvement along the axes, making displacement to core 2 a possibility.*

## 7.6 Biased LUCIE

One of the shortcomings of E-LUCIE is that it only considers *distance* to the nearest core when it evaluates a core's cost. The candidate set can contain many cores that are Pareto-optimal for only a few benchmarks. It is therefore possible that when a core is removed, a benchmark associated with the core is displaced to a core that was not originally Pareto-optimal for the benchmark. While this is not necessarily undesirable, it creates the possibility that as the number of cores in the set decreases, the selection drifts away from the Pareto frontier. The issue is illustrated in **figure 7.2**. If the square core is to be removed, E-LUCIE will displace the benchmark to the closest core, core 1. However, this increases both power consumption and execution time for the benchmark. In contrast, biased LUCIE (*BE-LUCIE*) is aware of the *direction* of the displacement, and will sometimes find that displacing the benchmark to core 2 is a better option. The details of the modified version of LUCIE are described below, followed by an example.

### 7.6.1 Definition

BE-LUCIE is shown in **algorithm 2**. It is identical to the basic version in **algorithm 1**, except that it uses a biased Euclidean distance function, **BE**, instead of the regular Euclidean distance function, **E**.

The biased distance function is Euclidean distance multiplied by a biasing factor. **BE** is defined in **equation 7.7**. The biasing factor is based on $\Delta P$ and $\Delta T$, the power

**Algorithm 2** BE-LUCIE (biased LUCIE) using default $\underline{\mathbf{C}}$ and biased distance $\underline{\mathbf{BE}}$

```
while |C| > N do
    for all c in C do
        C(c)                                   ▷ Equations 7.2, 7.6, 7.7
    end for
    c ← FINDMINCOSTCORE(C)
    remove c from C
    for all b in B_c do
        cm ← arg min_k BE(c, k, b)             ▷ Equations 7.7, 7.4
        append b to list for cm if not present
    end for
end while
```

and time changes that result from displacing a benchmark from a core. Recall that positive values of ΔP and ΔT signify an improvement. Since the cores follow a Pareto frontier, a situation where both are positive is unlikely. If, while LUCIE is running, the situation does arise where both ΔP and ΔT are positive, then this indicates that moving benchmark $b$ from $c_1$ to $c_2$ improves both power and execution speed. The biasing factor becomes less than one, reducing the cost of $c_1$ and making its removal more likely. If the move from $c_1$ to $c_2$ greatly reduces ΔP at a small detriment to ΔT, or vice-versa, then the biasing factor also becomes less than one and favors the move. This would improve the likelihood of a move to core 2 in **figure 7.2**. If ΔP and ΔT are both negative, then the biasing factor will be greater than one, and will increase the cost of the move. This would decrease the likelihood of the move to core 1 in **figure 7.2**. For most Pareto-optimal cores, the Δ values will have similar magnitudes but opposite signs, and the biasing factor's effect will be negligible.

$$\begin{aligned} \underline{\mathbf{D}}(c,b) &= \underline{\mathbf{BE}}(c,cm,b) \\ cm &= \arg\min_k \underline{\mathbf{BE}}(c,k,b) \end{aligned} \tag{7.6}$$

$$\underline{\mathbf{BE}}(c_1,c_2,b) \;=\; \underline{\mathbf{E}}(c_1,c_2,b) \;\times\; \Big(1 \;-\; \big(\Delta\mathrm{P}(c_1,c_2,b) \;+\; \Delta\mathrm{T}(c_1,c_2,b)\big)\Big) \tag{7.7}$$

The biasing factor has two purposes. First, since $\mathcal{C}$ contains Pareto-optimal cores for all benchmarks, the biasing factor discourages, but does not prevent, displacing

a given benchmark to a nearby core that is not Pareto-optimal for the benchmark. Second, the biasing factor also directs LUCIE away from extreme cores—cores that trade substantial increases in power for marginal performance increases, and cores that trade marginal power reductions for substantial performance reductions. I.e., the biasing factor helps avoid an unnecessarily broad *spread* (see **section 5.5.1** (p. 61)).

$\Delta$P and $\Delta$T are both normalized values that have no units, and they can therefore be summed. As more and more cores are removed, the gaps between remaining cores become larger, and it is possible for the biasing factor to become negative. There is no particular significance to a negative biasing factor. Since the costs of all cores are evaluated relative to each other, it is not even problematic if the costs of some cores become negative. Using a biasing factor that sometimes prefers further away cores over nearby cores may seem counterintuitive. However, most cores in a candidate set must eventually be removed, and selecting against cores that implement extreme power-performance trade-offs and against cores that are behind the Pareto frontier improves the quality of the set of remaining cores.

### 7.6.2 Example

**Figure 7.3** shows the progression of the BE-LUCIE algorithm as it removes cores. Notable differences between this example and the E-LUCIE plots in **figure 7.1** are the reduced number of low-affinity cores in the 128-core selection and the smaller spread in the final, 4-core selection. Both can be explained by the biasing factor. The cluster of low-affinity, high-power cores in the candidate set contains cores that should not be implemented under normal circumstances. These cores provide a speed improvement to very few benchmarks, the improvement is marginal, and is expensive in terms of power. The disproportionate power consumption of these cores causes the biasing factor to lower their costs, and they are quickly removed.

Similarly, the biasing factor reduces the spread of the selection. One must compare **figure 7.3** to **figure 7.1** to see the benefit of this. Of the two lowest-power cores selected by E-LUCIE, the slower one is substantially slower, but uses only marginally less power. It is unlikely that this core would be used sufficiently frequently to justify its inclusion. Of the two highest-power cores selected by E-LUCIE, the faster one is only marginally faster, but consumes considerably more power. It is unlikely that a power-constrained device will be able to use this core frequently. BE-LUCIE avoids

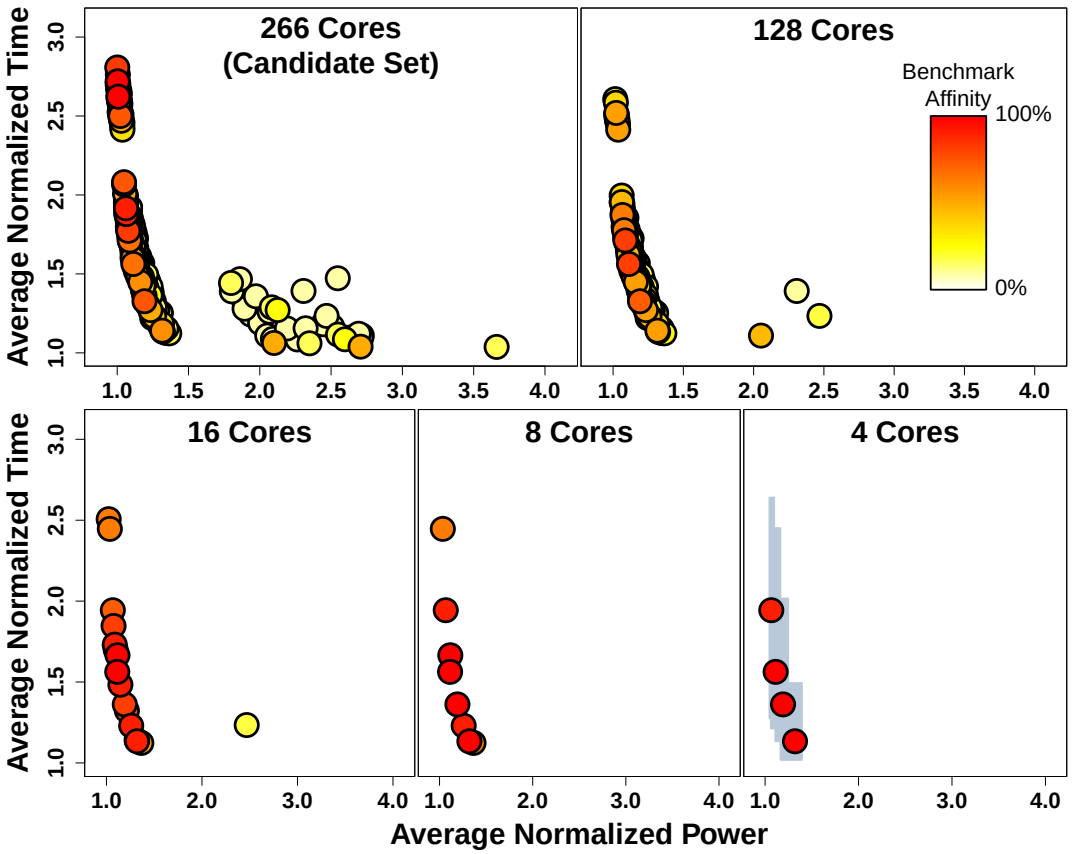


**Figure 7.3:** *BE-LUCIE is more effective than E-LUCIE at removing cores with low affinities and at removing extreme cores. Results are for* ***SPECint 2006****.*

these extreme cores. If extreme cores are required, then core pinning (**section 7.8**) and power PDFs (**section 7.9**) provide the designer with the means of including them.

## 7.7 Weighted-Biased LUCIE

A feature of both the basic and biased versions of LUCIE is how they handle outlier benchmarks. If a core in the candidate set has a very low affinity—if it is Pareto-optimal for very few benchmarks—then it will have a low cost and will be eliminated. This is the intended behavior, but the behavior can lead to benchmark *traveling*. If a benchmark is an outlier and its behavior is significantly different from that of other benchmarks, then it is possible for there to be a region of the candidate set with many cores that are Pareto-optimal for only the one benchmark. As each core is eliminated, the benchmark is displaced to a neighboring core. Since the entire region of the design space is useful to only one benchmark, the benchmark can "travel" across all the cores until they are all eliminated, and the benchmark loses access to the region of the space.

**Algorithm 3** WB-LUCIE (weighted-biased LUCIE) using $\underline{\mathbf{C}}$ with move counter and biased distance $\underline{\mathbf{BE}}$

**while** $|\mathcal{C}| > N$ **do**
  **for all** $c$ in $\mathcal{C}$ **do**
    $\underline{\mathbf{C}}(c)$ ▷ Equations 7.8, 7.6, 7.7
  **end for**
  $c \leftarrow$ FINDMINCOSTCORE($\mathcal{C}$)
  remove $c$ from $\mathcal{C}$
  **for all** $b$ in $\mathcal{B}_c$ **do**
    $cm \leftarrow \arg\min_k \underline{\mathbf{BE}}(c,k,b)$ ▷ Equations 7.7, 7.4
    append $b$ to list for $cm$ if not present
    $m_{cm,b} \leftarrow m_{cm,b} + m_{c,b} + 1$
  **end for**
**end while**

Other benchmarks are unaffected by this, but the one benchmark's runtime flexibility can be hampered substantially by the loss of an entire region of its Pareto-optimal frontier.

The traveling problem is an inevitable consequence of the LUCIE algorithm attempting to represent a candidate set of cores with a small selection. If, for example, four cores are selected, but there are 100 representative benchmarks, then it is unreasonable to reserve one core for the exclusive use of one benchmark. However, there can be benefits to adding awareness of the traveling problem into the LUCIE algorithm. The weighted-biased version of LUCIE, *WB-LUCIE* includes some "stickiness" to counteract traveling. The modified algorithm is defined below, followed by an example.

### 7.7.1 Definition

WB-LUCIE is shown in **algorithm 3**. It is identical to the biased version in **algorithm 2**, except that it uses the modified cost function in **equation 7.8** and it includes a *move counter*, $m_{c,b}$. Each benchmark, $b$, in the affinity list of each core, $c$ has an associated move counter, $m_{c,b}$. The move counter has an initial value of 0. When a core is removed and each benchmark associated with the core is displaced, then each benchmark's move counter is incremented and added to the destination core's move counter. If the benchmark was not already associated with the destination core,

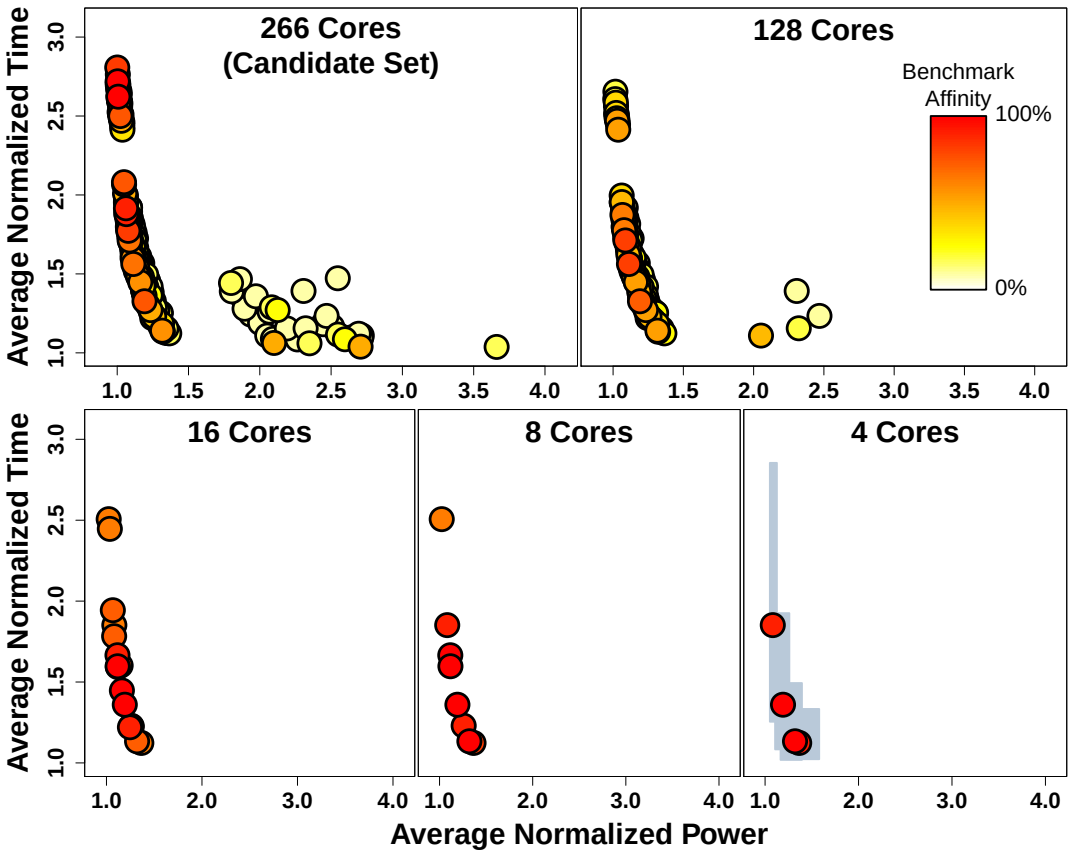


**Figure 7.4:** *WB-LUCIE places a stronger emphasis on per-benchmark specialization than BE-LUCIE, which leads to two high-performance cores. Results are for* ***SPECint 2006****.*

then the destination core's move counter for the benchmark is first initialized to 0. The move counter is used in the modified cost function, **equation 7.8**, to increase the displacement cost. If a benchmark has already been displaced many times, then its move counter will be large, and the cost of the core will be greater. This adds a measure of "stickiness" to the LUCIE algorithm. An outlier benchmark can only lose a limited number of cores before other benchmarks must also lose cores.

$$\underline{\mathbf{C}}(c) = \sum_{b}^{\mathcal{B}_c} \underline{\mathbf{D}}(c,b) \times (m_{c,b} + 1) \tag{7.8}$$

### 7.7.2 Example

**Figure 7.4** shows the progression of the WB-LUCIE algorithm. WB-LUCIE is visually similar to BE-LUCIE in **figure 7.3**. One notable difference is that in the 4-core selection, the weighted version selects two high-performance cores. This is because

benchmarks on all the high-performance, low-affinity cores converge to this region and bring along their large move counters. Note that the cores are plotted by average normalized power and time—two cores that appear to overlap in the averaged space do not necessarily overlap in each core's own power-performance space. A minor difference is the inclusion of a fourth high-power core in the selection of 128 cores. This is also the effect of the move counters, which have responded to traveling behavior. However, these high-power cores have such low affinities and such disproportionate power consumption that even with move counters, they are rightly eliminated.

## 7.8 Pinning Cores

By default, the LUCIE algorithm eliminates cores that are Pareto-optimal for only a few benchmarks, and cores that trade disproportionately large amounts of power for marginal performance improvements (or vice-versa). The underlying assumption is that these cores are a waste of silicon resources—cores that are optimal for only a few benchmarks will only rarely be required, and cores that consume large amounts of power will only rarely be usable in a power-limited device. In some cases, however, a designer might have good reason to force LUCIE to select a core that would normally be excluded. This is referred to as *core pinning*. A core, $c$ is pinned by setting its cost, $\underline{\mathbf{C}}(c)$, to infinity. LUCIE is run normally, benchmark affinities are calculated even for the pinned cores, but the pinned cores are never removed. Two use cases for core pinning are described below: maximizing best-case execution speed and incrementally designing processors. It should be noted that core pinning is possible because of the subtractive selection strategy employed by LUCIE. Pinning is not possible for some competing selection algorithms, like the Clustering strategy [71].

### 7.8.1 Maximization Example

LUCIE assumes that the power available to a program can vary, but some types of programs might be so important that the device will always make enough power available to run them as fast as possible (by, e.g., pausing other running programs, turning off radio interfaces, dimming the screen, etc.). For such a device, the designer may choose to pin a fast, high-power core. This can be demonstrated by using WB-LUCIE to select four cores for the SPECint 2006 benchmarks. In the first case, WB-LUCIE simply selects four cores. In the second case, a fast core (core #518) is pinned,

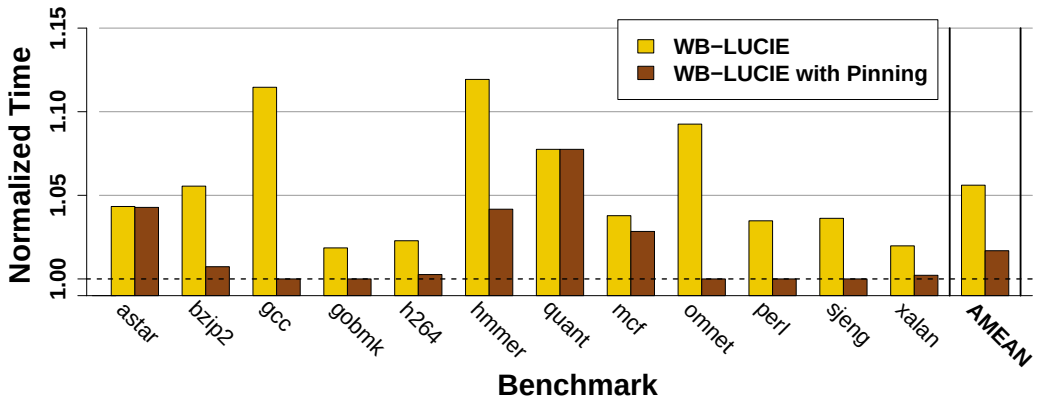


**Figure 7.5:** *Pinning a fast core increases speed for most benchmarks in situations where power is not limited. Averaged normalized time is 1.06 without pinning, 1.02 with pinning (smaller is better). Data is for* ***SPECint 2006*** *benchmarks.*

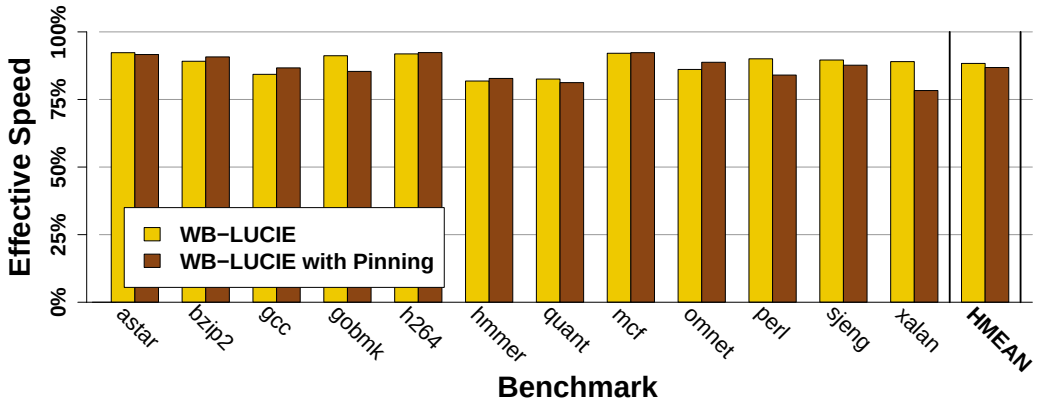


**Figure 7.6:** *Pinning a fast core slightly reduces the ES of benchmarks, because fast cores require more power and can be used less frequently. Average ES is 88% without pinning, 87% with pinning (larger is better). Data is for* ***SPECint 2006*** *benchmarks.*

and WB-LUCIE selects three more cores. The fast core is chosen because of its large caches and integer register file, and because it is Pareto-optimal for half of the SPEC benchmarks (most fast cores are optimal for fewer benchmarks). The minimum normalized execution times for the two selections are shown in **figure 7.5**. This is the best-case execution time when power is not limited. The average improvement from pinning is nearly 4%. The pinned core is not able to run *libquantum* faster than any of the cores normally selected by WB-LUCIE.

Pinning is not, however, an unqualified improvement, but a trade-off between best-case and average performance. Pinning a fast core can reduce the speed of a processor, because the fast core requires more power and can be used less frequently. **Figure 7.6** shows that for this example, pinning causes only a 1% reduction in effective speed, as

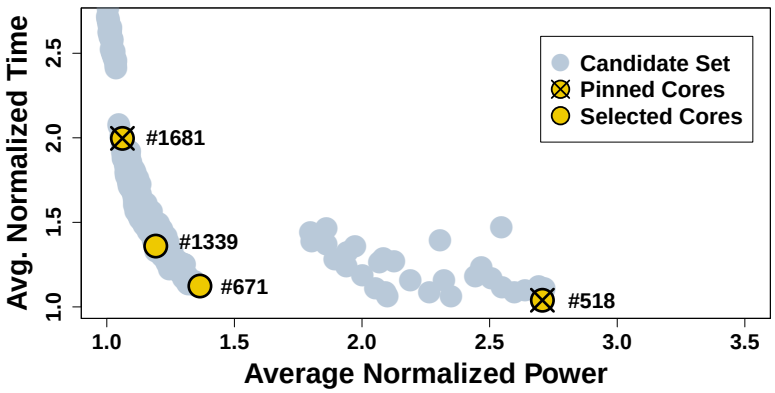


**Figure 7.7:** *When a slow and a fast core are pinned, WB-LUCIE selects cores in the middle. Cores are selected for* ***SPECint 2006*** *benchmarks.*

the selections of cores are not substantially different. The side effects of pinning will be more drastic if a core with a very low affinity is pinned, or if more core types are pinned.

### 7.8.2 Incremental Design Example

A second use of core pinning is when the designer has already implemented some cores (for, e.g., a previous product), and wishes to select additional cores to complement the existing ones. In this case, the designer simply pins the existing cores and runs LUCIE as normal. Two cores are pinned in **figure 7.7**, core #518 and core #1681. The performance difference between the pinned cores is approximately 2.0×, which is similar to the performance difference between core types in ARM's "big.LITTLE" configuration [69]. With the two cores pinned, WB-LUCIE selects core #671 as the third core. Adding core #671 leads to a speedup of 56% compared to the 2-core case, as measured by set overhead. If WB-LUCIE is allowed to select a fourth core, core #1339 is selected. This increases the speedup to 61% over the 2-core case.

## 7.9 Applying a Power PDF

In some circumstances, a designer might wish to guide the selection process without pinning specific cores. For example, the designer might know that real usage patterns emphasize a specific region of the design space. By using an available power PDF (probability density function), it is possible to increase the cost of cores in important regions of the space. If used, the PDF would be the same one described in

the runtime model (**section 5.4** (p. 58)) and used by metric weighting (**section 6.13** (p. 113)). LUCIE still selects cores for power flexibility and to balance the needs of all benchmarks, but it places a greater emphasis on the important regions.

The available power PDF is used as follows: The PDF applies a factor, $f(c,b)$, to the displacement cost function, $\underline{\mathbf{D}}(c,b)$, as shown in **equation 7.9**. The factor can be used with either the Euclidean distance function, $\underline{\mathbf{E}}$, or the biased Euclidean distance function, $\underline{\mathbf{BE}}$. The PDF modifies the cost of displacing core $c$ for each benchmark $b$. $f(c_i,b)$ is defined in **equation 7.10**. It is the integral of the PDF function for benchmark $b$ from the midpoint between core $c_i$ and its neighbor on the left to the midpoint between $c_i$ and its neighbor to the right. Cores are ordered by increasing power. The integration range for the first and last core begins at the lower-bound of the PDF and ends at the upper-bound of the PDF, respectively. Integrating with respect to a core's neighbors ensures that the PDF in a given region of the design space is divided among the cores in that region. As cores are removed, the remaining cores gain an increased benefit from the PDF. This prevents tight clusters of cores forming in regions of the design space with a high probability density.

$$\begin{aligned} \underline{\mathbf{D}}(c,b) &= f(c,b) \times \underline{\mathbf{BE}}(c,cm,b) \\ cm &= \arg\min_{k} \underline{\mathbf{BE}}(c,k,b) \end{aligned} \tag{7.9}$$

$$f(c_i,b) = \int_{l_1}^{l_2} \mathrm{PDF}_b$$

$$\begin{aligned} l_1 &= \frac{\mathrm{P_{raw}}(c_{i-1},b) + \mathrm{P_{raw}}(c_i,b)}{2} \\ l_2 &= \frac{\mathrm{P_{raw}}(c_i,b) + \mathrm{P_{raw}}(c_{i+1},b)}{2} \end{aligned} \tag{7.10}$$

The application of a power PDF is demonstrated in **figure 7.8**. The top plot shows five cores selected by BE-LUCIE for the entire EEMBC suite. The bottom plot shows an empirical distribution of CPU load data collected from a modern Android smartphone. I.e., it shows how likely the phone is to experience a given load. This previously unpublished load data was collected by Peter Henderson as part of an MSc project at the University of Edinburgh. It is simplistically assumed that the amount of power available to the CPU correlates with the load—if the load is low, it is because there is insufficient power to run a greater load. In reality, determining the source

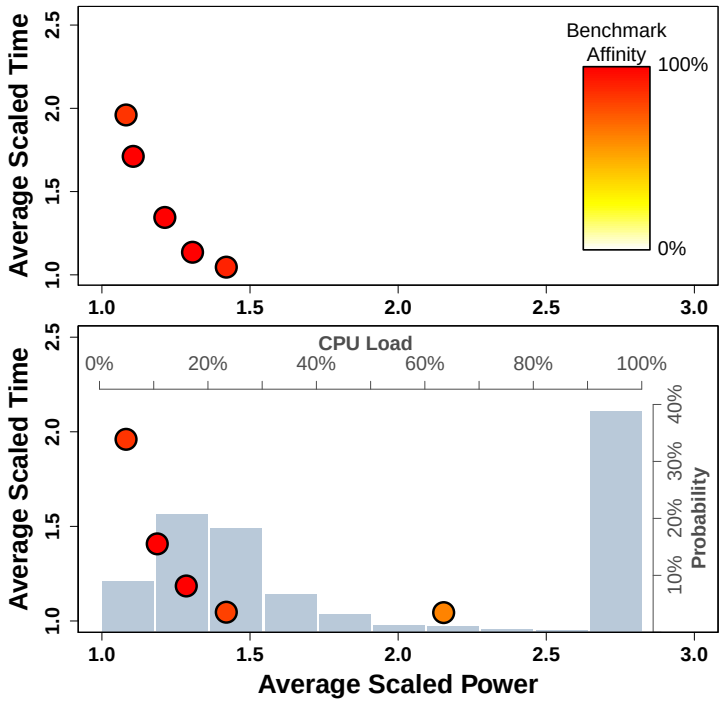


**Figure 7.8:** *(Top) Five cores selected by BE-LUCIE. (Bottom) Five cores selected by BE-LUCIE when the design space has empirical CPU load data applied to it (gray). Selection is for the complete set of 21* ***EEMBC*** *benchmarks.*

of CPU load is far more complicated. However, the load distribution is a reasonable example of the amount of power that a scheduler may make available to a task, and is sufficient for illustrating the application of a power PDF. The PDF is approximately scaled to the power axis using the range of power values in the candidate set.

The smartphone CPU spends much of its time at 0%-30% load, and also at 90%-100% load. Given the above assumption, this informs LUCIE that cores that can operate at below 30% of full power and at 90% of full power are more useful than other cores. The effect is that during selection, LUCIE discards the second lowest-power core, keeps the three medium cores, and introduces a high-power core. Note that the PDF only steers LUCIE—it does not completely dominate. The trade-off mechanisms in LUCIE still apply, and LUCIE still selects against cores with disproportionately high power for their performance.

## 7.10 Summary

The complexity of the selection problem lies in the fact that a small number of cores must be used to provide multiple power-performance points to tasks that can poten-

tially have very different behaviors. The LUCIE algorithm uses the candidate set of cores as the basis for determining the maximum theoretical diversity available to each benchmark in a representative set. LUCIE does not evaluate the average features of cores—cores are not considered to be "generally high-performance," or "generally low-power," etc. Instead, cores are evaluated based on their contribution to diversity as seen by each benchmark. A given core might be the fastest option for one benchmark, while offering only moderate performance to another benchmark with different microarchitectural requirements. As long as the core is Pareto-optimal (or nearly so) for both benchmarks and offers both benchmarks a unique power-performance point, it is considered useful.

This chapter has been limited to a subjective evaluation of the spread and uniformity produced by the three versions of the LUCIE algorithm. The next chapter performs a thorough, quantitative analysis on LUCIE and competing selection strategies.

# Chapter 8:

# LUCIE Evaluation & Metrics Demonstration

Once a processor designer has arrived at a set of heterogeneous cores, the set must be evaluated. Evaluation exists to determine the quality of the selection of cores, and, by extension, the quality of the algorithm that performed the selection. Evaluation itself is non-trivial—it requires a correct application and interpretation of metrics. This chapter evaluates LUCIE from **chapter 7** and two competing selection algorithms, and serves as a case study of the usage of the metrics defined in **chapter 6**.

## 8.1 Introduction

The problem of selecting a diverse set of heterogeneous cores is particularly relevant to mobile processors, as they must maximize performance for a range of tasks under a range of power budgets. This chapter evaluates the three versions of LUCIE from **chapter 7**, the *GA* selection strategy [134], and the *Clustering* selection strategy [71]. The three versions of LUCIE are basic LUCIE (E-LUCIE), which uses Euclidean distance; biased LUCIE (BE-LUCIE), which uses biased Euclidean distance; and weighted-biased LUCIE (WB-LUCIE), which uses biased distance and a move counter to weight cores.

The five algorithms are used to select a set of cores for the SPECint 2006 benchmark suite, and an independent set of cores for the EEMBC benchmark suite (see **section 4.3.1** (p. 41)). The majority of the chapter uses selections of four cores. This is within reach of current commercial technology, and it is a number commonly used in existing literature [96, 134]. The selection strategies select cores from the design space

of 3000 cores described in **section 4.3.2** (p. 42). The complete list of cores is included in **appendix A**. Cores are numbered from #1 to #3000.

The implementation details of each algorithm are presented first, followed by a qualitative section describing the selected cores. A section on summary metrics is included to demonstrate their limitations and the need for more descriptive metrics. This is followed by another qualitative section on the spread of the selected cores. The availability, uniformity, set overhead, and effective speed of the selections are then evaluated. The GA selection strategy only selects high-performance cores that have very low availability and are not suitable for mobile devices. This strategy is not considered after the availability section. A section on scalability evaluates gap overhead, generality, and monotonicity when the selection algorithms are used to select fewer or more than four cores. The final evaluation applies set overhead to a scenario where cores are selected using a non-flat power probability density function (PDF). The chapter ends with a concluding discussion on selection algorithms. The three versions of LUCIE are found to produce comparable results, and to consistently outperform the Clustering strategy.

## 8.2 Selection Algorithm Implementations

The analyses in this chapter evaluate up to five different core selection strategies. These are E-LUCIE, BE-LUCIE, WB-LUCIE, GA selection, and Clustering selection. The three versions of LUCIE were defined in **chapter 7**. The GA selection strategy by Navada et al. [134] and the Clustering selection strategy by Guevara et al. [71] represent the current state of the art in heterogeneous core selection. The two strategies were qualitatively critiqued in **section 5.7.3** (p. 70) and **section 5.7.4** (p. 71). This section describes the implementations of each strategy.

The three versions of LUCIE are implemented using a perl script. The version of LUCIE that should be used is selected with command line options.

The GA selection method is based on a genetic algorithm (GA) that combines both design space exploration and core selection into one step. The algorithm selects cores to optimize execution speed. Navada et al. [134] use a design space with over 13,000 cores. Since the example design space used in this chapter is much smaller at 3000 cores, the GA selection method is approximated with a hill-climbing search that optimizes speed. The search is implemented in R [145].

The Clustering selection algorithm is implemented in R using the default R implementation of k-means clustering. Cores are clustered based on the $\text{BIPS}^3/W$ efficiency metric. $\text{BIPS}^3/W$—billions of instructions per second per Watt—is inversely proportional to $\text{ED}^2$. The representative core from each cluster is selected to minimize the coefficient of variation (CoV) of $\text{BIPS}^3/W$. Guevara et al. [71] find these clustering and selection criteria to lead to the best results.

When running on a modern Intel processor, both LUCIE and the Clustering strategy require only a few seconds to select cores from a candidate set. It should be noted, however, that since the Clustering strategy is based on k-means clustering, it is non-deterministic. K-means can sometimes fail to find good clusters, in which case the Clustering strategy will fail to select a diverse set of cores. The GA strategy runs in under two hours. It is much slower both because it traverses the entire design space rather than just the candidate set, and because R is not conducive to tasks with a large proportion of control flow operations.

## 8.3 Selected Cores

This section contains a qualitative analysis of the features of the selection algorithms. The differences between the three versions of the LUCIE algorithm have already been described in **chapter 7**, so this section only considers WB-LUCIE, the GA strategy, and the Clustering strategy. While the specific cores selected by the algorithms are different for the SPEC and EEMBC suites due to differences in benchmark features, the underlying behaviors of the algorithms are independent of the benchmark suite. The focus will be on the cores selected for the SPEC suite; the discussion is largely identical for the EEMBC suite.

**Table 8.1** lists the four cores selected for the SPECint 2006 benchmark suite by each of the five selection strategies, and **table 8.2** lists the cores selected for the combined EEMBC suite. **Figure 8.1** and **figure 8.2** plot the selections, omitting the basic and biased versions of LUCIE for visual clarity. The behavioral differences of the three algorithms are obvious from inspection of **figure 8.1**. The GA strategy optimizes for performance and selects four fast, high-power cores. The Clustering strategy divides the candidate set into four regions and selects a core from each region. The LUCIE strategy avoids extremes and selects cores around the knee of the candidate set. As noted in **section 7.7.2** (p. 131), cores that appear close together in the average space

| SPECint 2006 Cores | | | | |
|---|---|---|---|---|
| **E-LUCIE Selection** | #142 | #1288 | #1449 | #518 |
| **BE-LUCIE Selection** | #1181 | #2417 | #1631 | #1449 |
| **WB-LUCIE Selection** | #2761 | #1339 | #1449 | #671 |
| **Clustering Selection** | #1981 | #2288 | #2145 | #319 |
| **GA Selection** | #1165 | #518 | #969 | #319 |

**Table 8.1:** *Four cores selected by each algorithm for the **SPECint 2006** benchmark suite. Cores are ordered from low to high power. Core configurations are in **table 8.3**, **table 8.4**, **table 8.5**, and **appendix A**.*

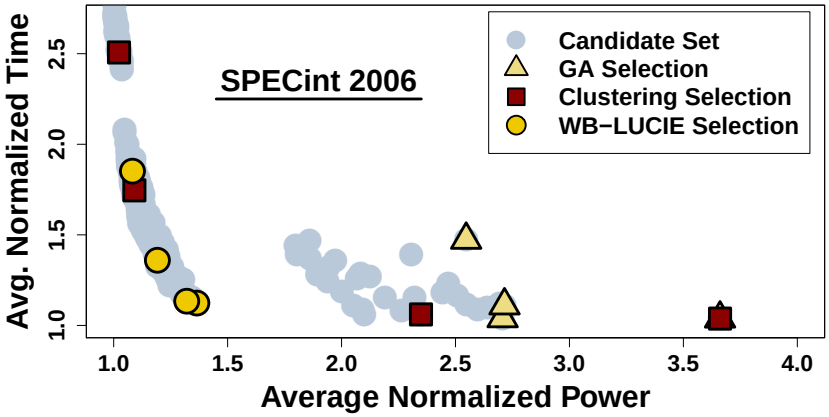


**Figure 8.1:** *Four cores selected by the GA strategy, the Clustering strategy, and the WB-LUCIE strategy for the **SPECint 2006** suite. Configurations of the cores are shown in **table 8.3**, **table 8.4**, and **table 8.5**. Note that the GA and Clustering strategies select the same highest-power core.*

are not necessarily close together in any benchmark's individual power-performance space.

All of the selection strategies are oblivious to how the cores are implemented; they only rely on measurable features of cores. The GA strategy uses execution speed, the Clustering strategy uses efficiency, and LUCIE uses a power-performance trade-off. However, studying the microarchitectural features of the cores selected by each algorithm leads to a greater insight into the algorithms' behaviors. In the example design space, power and performance are primarily controlled by the data cache (d-cache) and the integer register file (IR). Both of these structures are key to extracting instruction level parallelism (ILP), but their size and complexity also makes them significant consumers of power. At larger sizes, these structures become particularly

| EEMBC Cores | | | | |
|---|---|---|---|---|
| **E-LUCIE Selection** | #142 | #1546 | #1297 | #161 |
| **BE-LUCIE Selection** | #156 | #1056 | #2416 | #671 |
| **WB-LUCIE Selection** | #1056 | #821 | #1766 | #2525 |
| **Clustering Selection** | #2836 | #671 | #95 | #1984 |
| **GA Selection** | #2503 | #518 | #732 | #319 |

**Table 8.2:** *Four cores selected by each algorithm for the **EEMBC** benchmark suite. Cores are ordered from low to high power. Core configurations are in **appendix A**.*

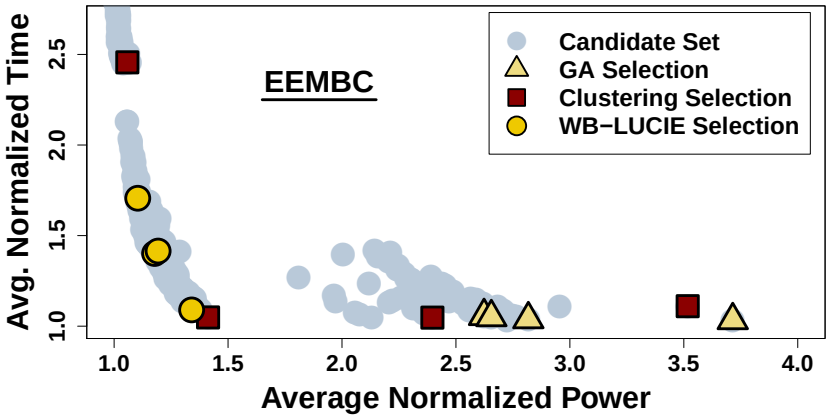


**Figure 8.2:** *Four cores selected by the GA strategy, the Clustering strategy, and the WB-LUCIE strategy for the **EEMBC** suite. Cores are listed in **table 8.2**.*

expensive due to the additional complexity required to meet timing. The instruction cache (i-cache) is not nearly as significant as the data cache, since the i-cache has only one hardware port compared to the two in the d-cache. Floating point registers are not as significant as integer registers, because the benchmarks perform few floating point operations. After the d-cache and IR, the queues also have a significant effect on power and performance. While queues consume little power themselves, they enable greater ILP and greater power consumption throughout the core. The branch predictor has little to no impact on benchmark performance. This has also been noted elsewhere [172], and may be due to the gem5 implementation of branching logic.

The differences between the algorithms are most obvious in the selected cores' d-cache sizes (DS) and IR sizes. The microarchitectural parameters for the four SPEC cores selected by the WB-LUCIE strategy, the Clustering strategy, and the GA strategy are listed in **table 8.3**, **table 8.4**, and **table 8.5**, respectively. Recall that the LUCIE strategy approximates the power-performance Pareto-optimal frontier for all

**WB-LUCIE Core Configurations**

(SPECint 2006)

| | Caches | | | | Registers | | Queues | | | | Branch Predictor | | | | | | | BTB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Core** | **DS** | **DW** | **IS** | **IW** | **IR** | **FPR** | **IQ** | **LQ** | **SQ** | **ROB** | **GCB** | **GE** | **LCB** | **LHB** | **LHE** | **CCB** | **CE** | **BE** | **BT** |
| **#2761** | 16kB | 1 | 8kB | 1 | 96 | 128 | 16 | 16 | 16 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 18 |
| **#1339** | 16kB | 2 | 32kB | 4 | 96 | 128 | 16 | 64 | 16 | 128 | 3 | $2^{14}$ | 1 | 11 | $2^{9}$ | 1 | $2^{14}$ | $2^{10}$ | 20 |
| **#1449** | 32kB | 2 | 64kB | 4 | 96 | 256 | 32 | 32 | 64 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 16 |
| **#671** | 32kB | 2 | 32kB | 4 | 128 | 96 | 64 | 32 | 32 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 |

**Table 8.3:** *Configurations of the four cores selected by the WB-LUCIE strategy for the **SPECint 2006** benchmark suite. Cores are ordered by increasing average normalized power and plotted in **figure 8.1**. The design space parameters are listed in **table 4.3** (p. 43).*

**Clustering Core Configurations**

(SPECint 2006)

| | Caches | | | | Registers | | Queues | | | | Branch Predictor | | | | | | | BTB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Core** | **DS** | **DW** | **IS** | **IW** | **IR** | **FPR** | **IQ** | **LQ** | **SQ** | **ROB** | **GCB** | **GE** | **LCB** | **LHB** | **LHE** | **CCB** | **CE** | **BE** | **BT** |
| **#1981** | 16kB | 2 | 64kB | 4 | 50 | 128 | 16 | 16 | 16 | 32 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 |
| **#2288** | 16kB | 1 | 64kB | 1 | 64 | 128 | 16 | 32 | 32 | 16 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 20 |
| **#2145** | 32kB | 1 | 64kB | 2 | 256 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 18 |
| **#319** | 64kB | 1 | 32kB | 4 | 256 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 |

**Table 8.4:** *Configurations of the four cores selected by the Clustering strategy for the **SPECint 2006** benchmark suite. Cores are ordered by increasing average normalized power and plotted in **figure 8.1**. The design space parameters are listed in **table 4.3** (p. 43).*

**GA Core Configurations**

(SPECint 2006)

| | Caches | | | | Registers | | Queues | | | | Branch Predictor | | | | | | | BTB | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Core** | **DS** | **DW** | **IS** | **IW** | **IR** | **FPR** | **IQ** | **LQ** | **SQ** | **ROB** | **GCB** | **GE** | **LCB** | **LHB** | **LHE** | **CCB** | **CE** | **BE** | **BT** |
| **#1165** | 16kB | 4 | 4kB | 1 | 256 | 128 | 64 | 64 | 64 | 128 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 16 |
| **#518** | 64kB | 2 | 64kB | 2 | 128 | 256 | 32 | 64 | 32 | 64 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 |
| **#969** | 64kB | 4 | 32kB | 1 | 128 | 128 | 32 | 64 | 64 | 128 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 16 |
| **#319** | 64kB | 1 | 32kB | 4 | 256 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 |

**Table 8.5:** *Configurations of the four cores selected by the GA strategy for the **SPECint 2006** benchmark suite. Cores are ordered by increasing average normalized power and plotted in **figure 8.1**. The design space parameters are listed in **table 4.3** (p. 43).*

benchmarks using just a few cores. In contrast, the Clustering strategy clusters cores by efficiency, and then selects from each cluster the core that is most consistent (has the lowest coefficient of variation, CoV) for efficiency across all benchmarks. The crucial limitation of the Clustering approach is that maximizing consistency is equivalent to taking the lowest common denominator—there is no scope for taking advantage of behavioral differences among benchmarks if cores must have similar efficiency for all benchmarks. For example, the slowest core selected by the Clustering method has 50 IR entries, whereas none of the cores selected by WB-LUCIE have fewer than 96. For the IR, the power difference between 50 and 96 entries is insignificant. The Clustering method uses a core with 50 IR entries because this creates an artificial bottleneck that keeps all benchmarks at a certain power and performance level. A larger IR would accelerate some benchmarks more than others, thereby decreasing consistency. Similar behavior can be observed with the largest core selected by the Clustering method, which contains both a 64kB d-cache and a 256-entry IR. None of the cores selected by WB-LUCIE contain a data cache or integer register file this large. These over-sized structures ensure that this largest core consistently consumes substantial power for all benchmarks, even though some benchmarks do not benefit from such large structures.

The GA strategy exacerbates the problems of the Clustering strategy by selecting only consistently fast cores. Of the four selected cores, all have large or very large IRs, and three have large d-caches. Even the one core with a small d-cache, #1165, consumes large amounts of power because of the complexity necessitated by the highly associative d-cache (DW) and the IR size. The selections are guaranteed to consume a disproportionate amount of power, and the lack of low-power cores precludes runtime flexibility.

## 8.4 Summary Metrics

Before proceeding to analyze the selections of cores with the metrics presented in **chapter 6**, it is necessary to confirm that existing metrics cannot accurately evaluate the selections. A major motivator for the metrics and the LUCIE algorithm is the inability of summary metrics to evaluate the diversity of a selection of cores. An intuitive description of this problem is in **section 5.6.2** (p. 64); the following demonstrates the problem with the example dataset.

The GA selections in **figure 8.1** and **figure 8.2** show that selecting cores for a single metric—performance—leads to a tightly clustered selection rather than a diverse

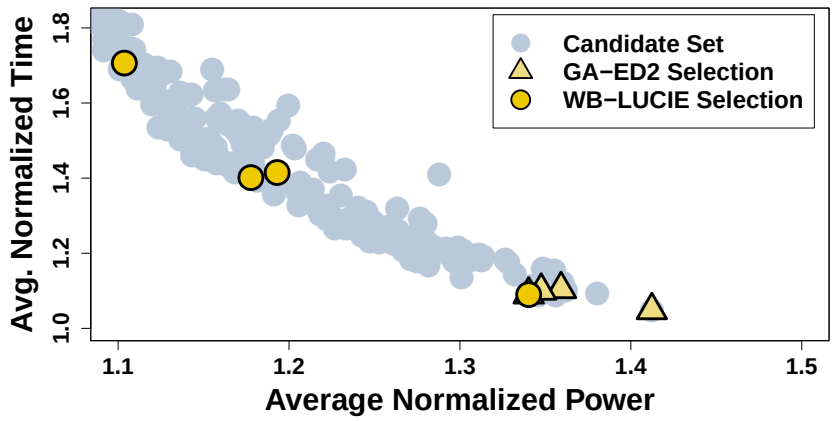


**Figure 8.3:** *Cores selected for $ED^2$ cluster in one region of the design space, whereas cores selected by LUCIE offer diversity. Results are shown for the **EEMBC** suite.*

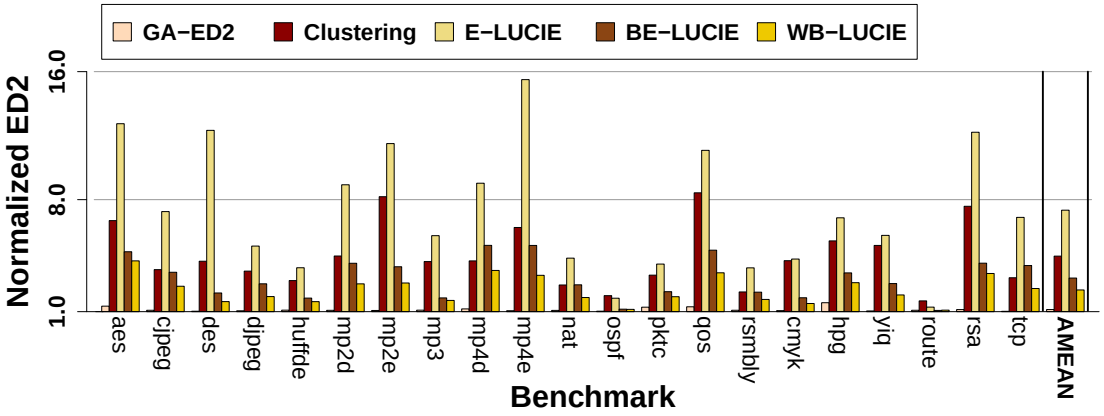


**Figure 8.4:** *Average normalized $ED^2$ for each selection of four cores. When $ED^2$ is optimized (GA-$ED^2$), the average $ED^2$ across all benchmarks is 1.13. For the other selections, average $ED^2$ values are 4.47, 7.34, 3.10, and 2.36 respectively. Results are for the **EEMBC** suite. Smaller values are better.*

selection. Selecting cores for a summary metric that attempts to balance competing metrics does not correct this. **Figure 8.3** shows the four cores that the GA strategy selects when it optimizes the $ED^2$ efficiency metric (GA-$ED^2$) instead of performance. The GA-$ED^2$ cores offer very little runtime flexibility compared to the diverse WB-LUCIE selection. Diversity requires a range of cores, but only one point in the design space minimizes $ED^2$.

**Figure 8.4** shows normalized $ED^2$ for each EEMBC benchmark averaged across the four cores in each selection. Intuitively, the diversity of the GA-$ED^2$ selection is considerably worse than the diversity of the other selections, but the $ED^2$ of the GA-$ED^2$ selection is consistently better. It can be concluded that optimizing for $ED^2$

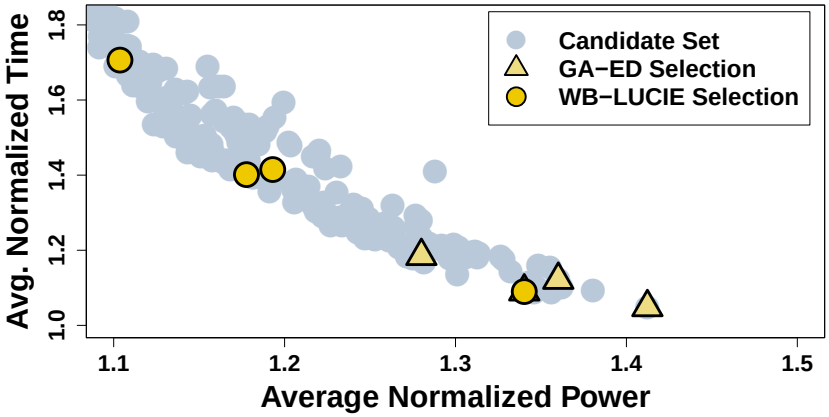


**Figure 8.5:** *Cores selected for ED cluster in one region of the design space, whereas cores selected by LUCIE offer diversity. Results are shown for the* ***EEMBC*** *suite.*

does not lead to the diverse selections required by mobile devices, and that diverse selections are unlikely to optimize $ED^2$.

The issue is not specific to $ED^2$, but applies to all summary metrics—using a different metric simply shifts the optimal point in the design space. **Figure 8.5** shows four cores selected by the GA strategy to optimize ED, the energy-delay product. Reducing the weight of the delay term shifts the selection of cores, but the cores still provide only a narrow range of power-performance points. It is clear that summary metrics are generally not sufficiently informative for evaluating heterogeneous processors.

## 8.5 Spread

The spread of a selection of cores is a qualitative feature loosely related to diversity. As described in **section 5.5.1** (p. 61), a flexible selection of cores must have some spread, but maximally spread out cores do not necessarily provide maximal flexibility.

**Figure 8.6** shows the spread of power values achieved by each algorithm for the SPEC 2006 integer suite. The chart follows directly from the descriptions of each selection strategy in **section 8.3** above and in **chapter 7**. The cores selected by the GA strategy always consume large amounts of power. The Clustering strategy offers the maximal amount of spread, since cores are selected for consistency—the lowest-power core contains bottlenecks that prevent power consumption, and the highest-power core contains large structures that always consume large amounts of power. E-LUCIE has a comparable spread to the Clustering strategy, but it avoids the highest-power cores. BE-LUCIE and WB-LUCIE have much lower maximum power consumptions, since

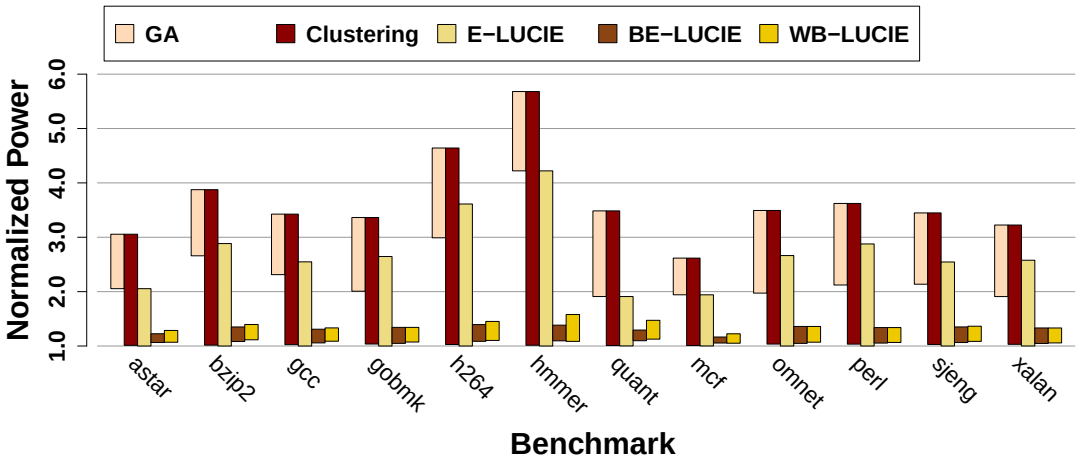


**Figure 8.6:** *Spread of normalized power values for **SPECint 2006** benchmarks on the five selections of cores. Smaller values are better. Cores for each selection are listed in **table 8.1**.*

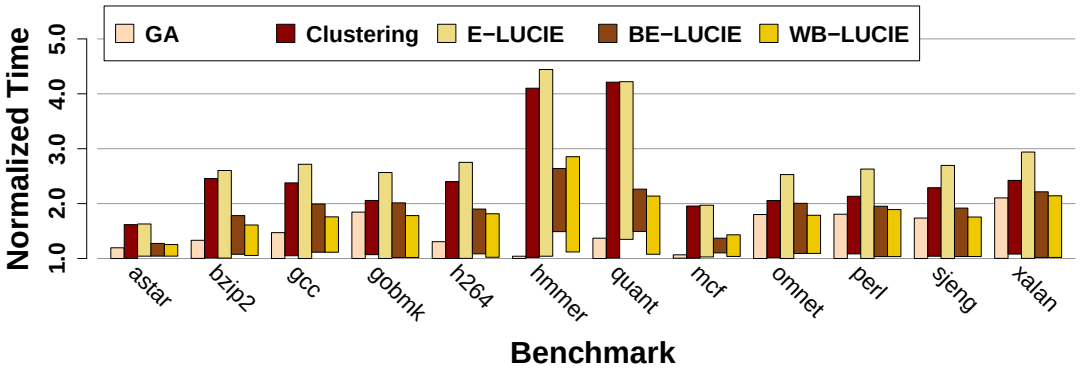


**Figure 8.7:** *Spread of normalized time values for **SPECint 2006** benchmarks on the five selections of cores. Smaller values are better. Cores for each selection are listed in **table 8.1**.*

they use a biasing factor that selects against extremely high-power cores. WB-LUCIE has a slightly higher maximum power owing to its use of a move counter—the large number of low-affinity cores in the design space pulls the maximum power of WB-LUCIE slightly higher than that of BE-LUCIE. All versions of LUCIE along with the Clustering strategy are able to operate at very low levels of power, as is often required by mobile devices.

Despite the small spread and low maximum power of the LUCIE selections, the LUCIE algorithm provides substantial runtime flexibility. The only justification for high-power cores is fast execution. **Figure 8.7** shows the spread of the five selections along the time axis. It can be seen that for most benchmarks, the cores selected by

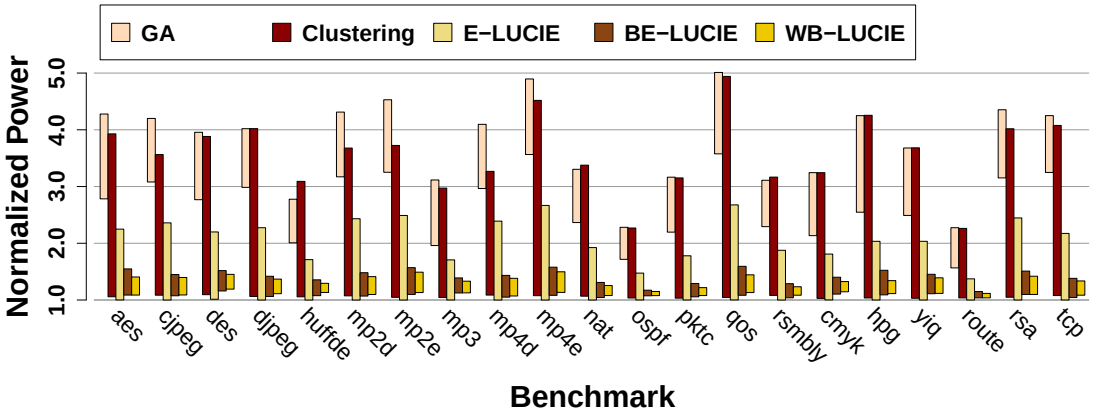


**Figure 8.8:** *Spread of normalized power values for **EEMBC** benchmarks on the five selections of cores. Smaller values are better. Cores for each selection are listed in **table 8.2**.*

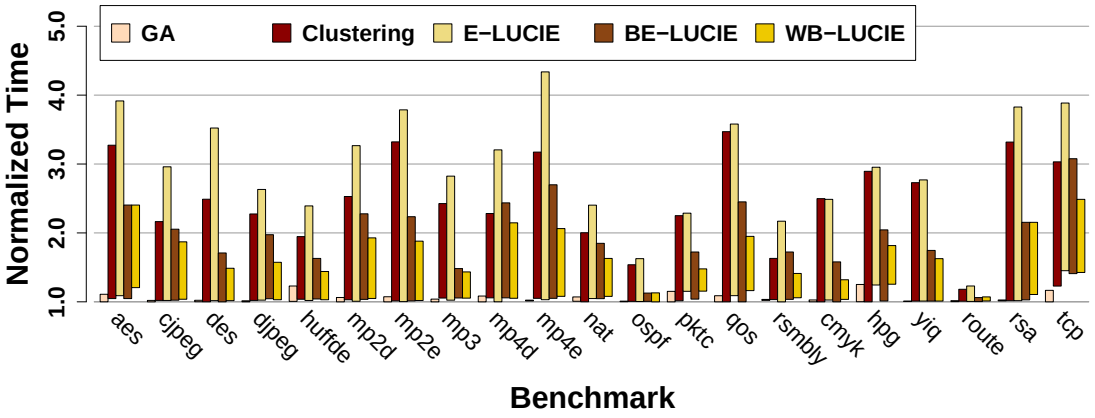


**Figure 8.9:** *Spread of normalized time values for **EEMBC** benchmarks on the five selections of cores. Smaller values are better. Cores for each selection are listed in **table 8.2**.*

LUCIE offer maximum performance that is nearly indistinguishable from that of the other selection strategies. Exceptions are the *hmmer* and *libquantum* benchmarks, where the fastest core selected by BE-LUCIE is noticeably slower than the fastest cores of other selections. The move counter in WB-LUCIE remedies this problem.

Power and performance spreads for the EEMBC benchmark suite are shown in **figure 8.8** and **figure 8.9** respectively. The selection behavior evident for the SPEC benchmarks is also evident for EEMBC benchmarks: The LUCIE selections have a narrower power range and a lower maximum power, but the effect on maximum performance is small. There is, however, a greater effect from LUCIE on maximum performance for the EEMBC suite than the SPEC suite. This is because the EEMBC

suite contains more benchmarks—21 compared to the 12 in SPEC—which makes it more difficult to select cores that are applicable to all benchmarks. The *tcp* benchmark in particular is an outlier, and only the GA strategy with its four high-performance cores can afford to select a core to maximize *tcp* performance.

These figures show qualitatively that while some amount of spread is required, maximizing spread is not necessary. An analysis of spread is helpful for understanding the trade-off between power and performance in a selection of cores. Spread does not, however, translate directly into performance. For example, a selection of cores with inferior spread may perform better than a selection with superior spread if there is not enough power to use the faster cores in the latter selection. The next sections perform quantitative analysis to better understand features of selection algorithms and selections of cores.

## 8.6 Availability

The availability, Av, of a selection of cores is the likelihood that at least one core in the selection can be used. Av is defined in **section 6.9** (p. 100). Availability is evaluated over the range of power values from the power of the lowest-power benchmark-core combination in the design space to the power of the highest-power benchmark-core combination of any of the five selections. The power PDF is assumed to be flat, and is assumed to be zero outside the evaluation range (see **section 5.4** (p. 58) and **section 6.13** (p. 113)).

**Figure 8.10** shows the availability of the selections for the SPEC suite. The GA selection has an average availability of 71%. Highest availability is for *mcf* at 81%; lowest availability is for *hmmer* at 31%. The other four selection strategies have nearly indistinguishable availabilities, with the average availability ranging from 97% to 99%. BE-LUCIE and WB-LUCIE have marginally lower availabilities than the Clustering and E-LUCIE strategies, because the former use a biased distance metric to avoid disproportionately low-power cores.

The availability comparison is similar for the EEMBC suite, as shown in **figure 8.11**. The Clustering and LUCIE strategies all have an average availability in the 97%–99% range. The availability of the GA selection is even lower for EEMBC than for SPEC, at an average of 59%.

The Av metric shows that the Clustering and LUCIE strategies lead to comparable, highly available selections. In contrast, the GA strategy makes selections of cores

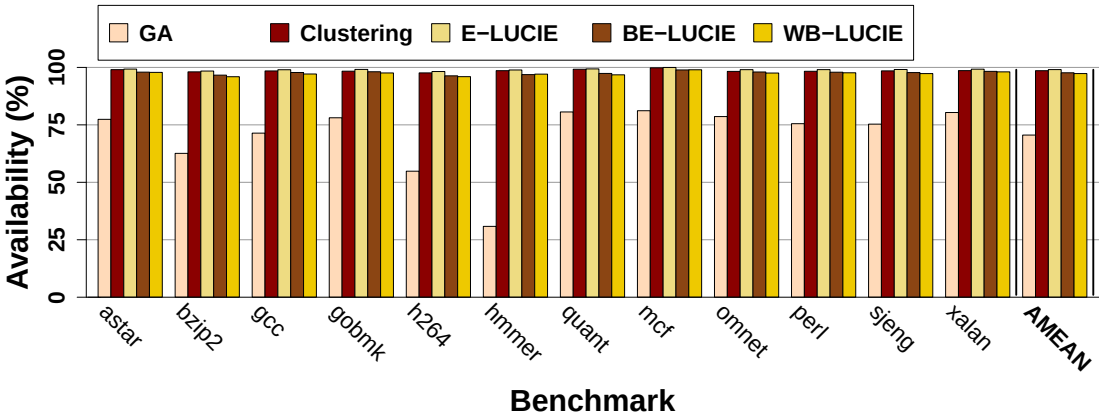


**Figure 8.10:** *Availability of the five selections of cores for the **SPECint 2006** suite. Average availabilities for the selections are 71%, 99%, 99%, 98%, 97%, respectively. Larger values are better.*

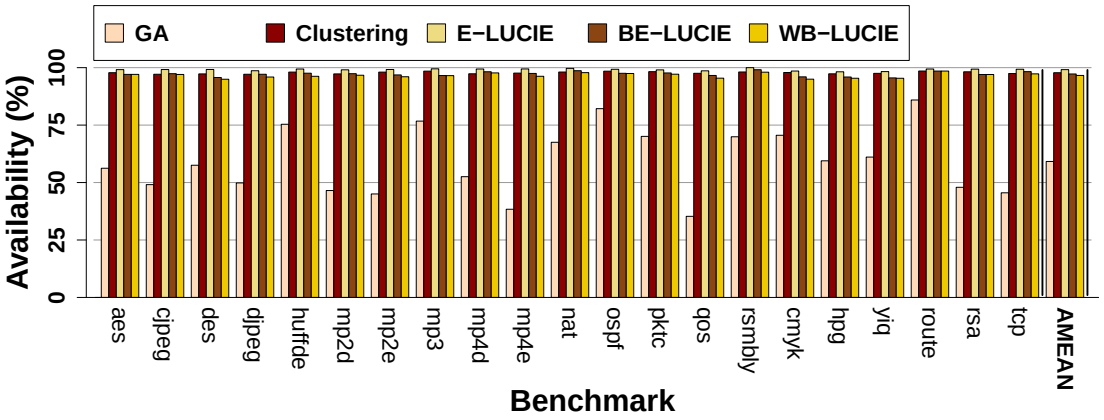


**Figure 8.11:** *Availability of the five selections of cores for the **EEMBC** suite. Average availabilities for the selections are 59%, 98%, 99%, 97%, 97%, respectively. Larger values are better.*

that are unusable under any but the most optimistic power budgets. The GA strategy cannot select cores that enable runtime flexibility in mobile devices. Consequently, the remaining evaluations in this chapter do not consider GA selection.

## 8.7 Localized Non-Uniformity

Localized non-uniformity, כ, evaluates how evenly cores are distributed. It is defined in **section 6.6.3** (p. 84). The non-uniformity of each selection-benchmark combination is evaluated over its individual power-performance range. The lower-bound of the range ($R_{min}$ in **equation 6.1** (p. 85)) is based on the power of the lowest-power core

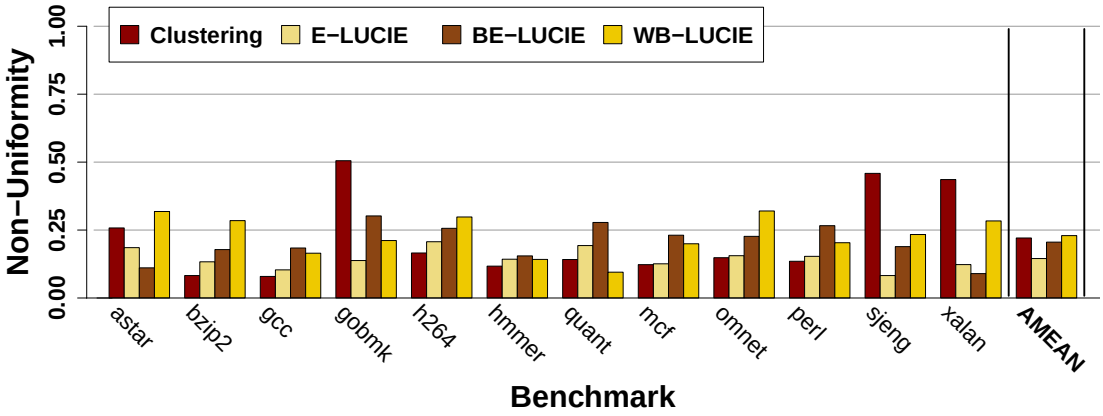


**Figure 8.12:** *Localized non-uniformity for the* ***SPECint 2006*** *suite. Average ↄ values are* 0.22, 0.15, 0.21, 0.23, *respectively. Smaller values are better.*

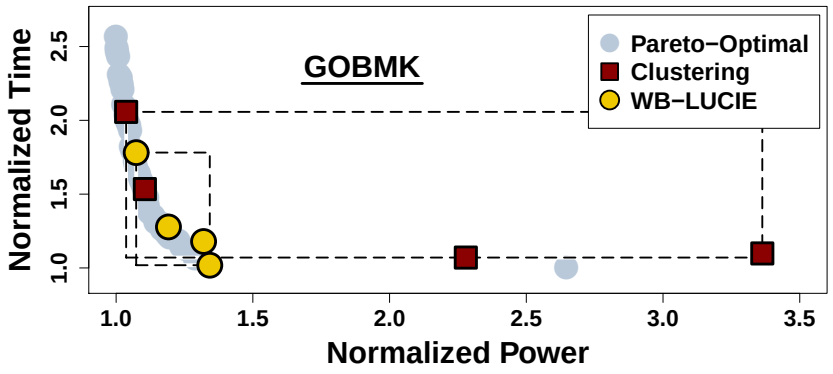


**Figure 8.13:** *Cores selected for the* ***SPECint 2006*** *suite by the Clustering and WB-LUCIE strategies, shown for the* gobmk *benchmark. Dashed boxes illustrate the bounding boxes of the selections. The highest-power core from the Clustering strategy is not Pareto-optimal for* gobmk.

and the speed of the slowest core in the given selection. The upper-bound ($R_{max}$ in **equation 6.1** (p. 85)) is based on the highest-power core and the fastest core. I.e., the range is computed using a bounding box.

**Figure 8.12** shows localized non-uniformity for SPEC benchmarks. The four algorithms provide comparable uniformity to the benchmarks. E-LUCIE has the best average ↄ at 0.15; the other two versions of LUCIE sacrifice some uniformity to avoid extreme cores. Based on its clustering and variation minimization approach, one would expect the Clustering strategy to have particularly low non-uniformity. This is generally the case, but **figure 8.12** shows that the Clustering selection has three outliers: *gobmk*, *sjeng*, and *xalancbmk*. **Figure 8.13** illustrates the source of these outliers with the *gobmk* benchmark. The most powerful core selected by the Clustering

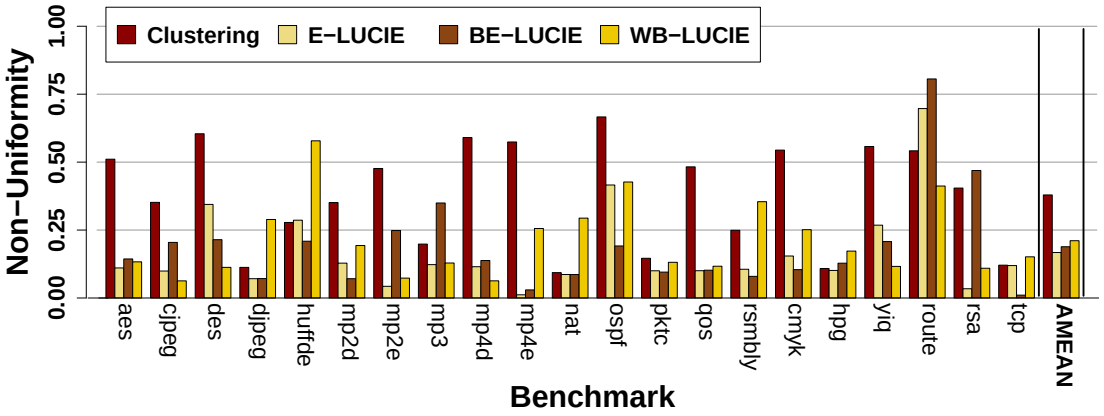


**Figure 8.14:** *Localized non-uniformity for the **EEMBC** suite. Average ↄ values are* 0.38*,* 0.17*,* 0.19*,* 0.21*, respectively. Smaller values are better.*

strategy is marginally slower than the second most powerful core for *gobmk*, and is therefore not Pareto-optimal. The range over which ↄ is calculated is determined by the bounding box of the selected cores, but ↄ is only calculated using Pareto-optimal cores (since other cores should not be used). As a result, the selected cores offer uneven coverage of the range. The WB-LUCIE selection similarly only has three cores that are Paret-optimal for *gobmk*. However, these three cores are evenly distributed and provide even coverage of the space.

**Figure 8.14** shows localized non-uniformity for EEMBC benchmarks. Selecting cores for the larger EEMBC suite is more difficult than selecting for the SPEC suite, and as a result, there are more outliers. The Clustering strategy in particular has severely degraded uniformity, as the highest-power core it selects only benefits few benchmarks. Despite the outliers, the average non-uniformity values of LUCIE selections degrade only marginally compared to the non-uniformity values for SPEC selections. The uniformity of the WB-LUCIE selection is worst for the *huffde* benchmark. For this selection, *huffde* suffers from the opposite problem as the Clustering selection—as can be seen in **figure 8.15**, the slowest core does not offer lower power than the second-slowest core. Furthermore, two of the other cores have very similar behavior and are partially redundant.

The ↄ metric helps identify cases where some benchmarks receive less of the benefits of heterogeneity than others, and cases where selected cores are (partially) redundant. For the SPEC suite, ↄ shows that the Clustering algorithm produces uniformity comparable to that produced by LUCIE. For the EEMBC suite, however, ↄ demonstrates that the strategy of selecting consistently high-power cores employed

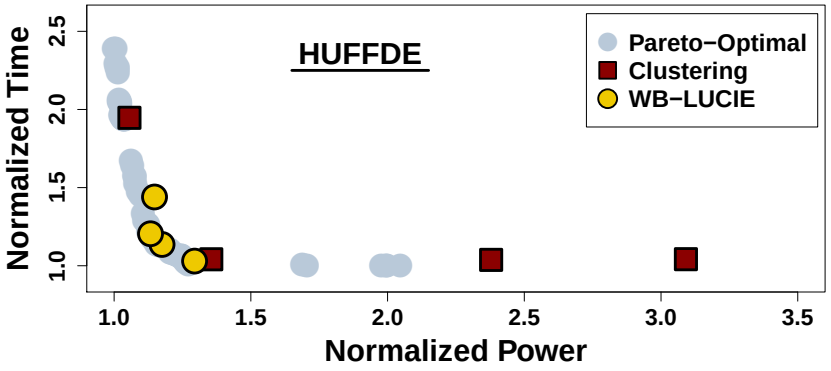


**Figure 8.15:** *Cores selected for the* ***EEMBC*** *suite by the Clustering and WB-LUCIE strategies, shown for the* huffde *benchmark. The slowest core selected by WB-LUCIE is not Pareto-optimal for* huffde*, increasing* ↄ*.*

by the Clustering algorithm begins to break down. Uniformity deteriorates because disproportionately high-power cores are selected.

## 8.8 Set Overhead

Set overhead, SO, measures the relative slowdown caused by using the "wrong" set of cores. It is defined in **section 6.8** (p. 93). This section evaluates set overhead for the Clustering, BE-LUCIE and WB-LUCIE selections using E-LUCIE as the baseline. For this analysis, the available power PDF is assumed to be flat, and the maximum amount of power ever available is taken to be the power of the highest-power benchmark-core combination of any of the four selections. Since the GA selection is not included, its cores are not used to determine maximum power.

**Figure 8.16** shows SO for the SPEC suite. On average, cores selected by the Clustering strategy are 11% slower than cores selected by E-LUCIE. Cores selected by BE-LUCIE are an average 3% slower than cores selected by E-LUCIE. WB-LUCIE provides a 3% speedup over E-LUCIE. The selections are particularly interesting for the *libquantum* benchmark, as the Clustering and E-LUCIE selections have nearly identical behaviors (SO $< 0.2\%$), while WB-LUCIE is 26% faster than E-LUCIE. **Figure 8.17** shows the cores selected by E-LUCIE, WB-LUCIE, and the Clustering strategy for *libquantum*. The two lowest-power cores selected by E-LUCIE and the Clustering strategy are almost identical. While the two highest-power cores in the E-LUCIE selection are slower than the two highest-power cores in the Clustering selection, the former also require much less power. The overall result is that for

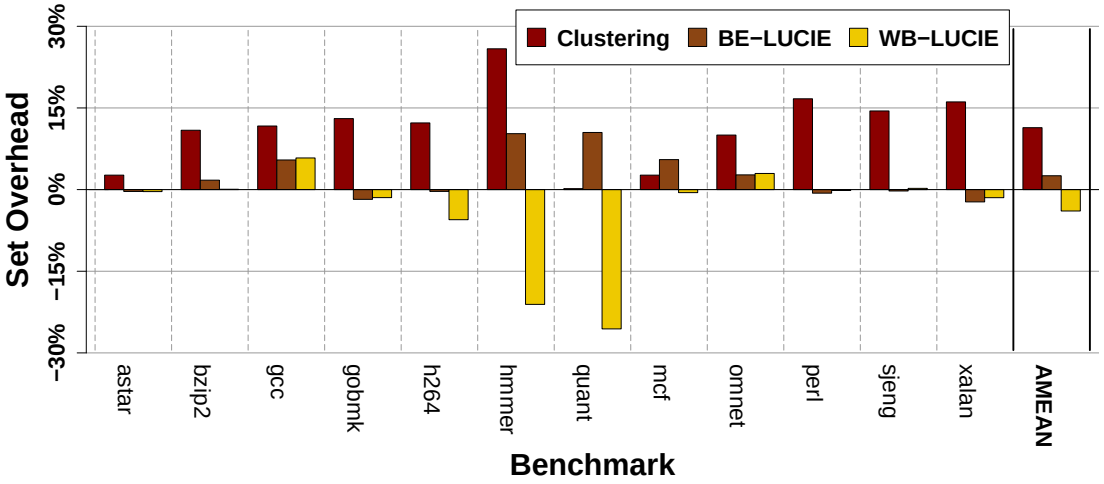


**Figure 8.16:** *Set overhead relative to the E-LUCIE selection for the* ***SPECint 2006*** *suite. Average SO values are 11%, 3%, and -4%, respectively. Smaller values are better.*

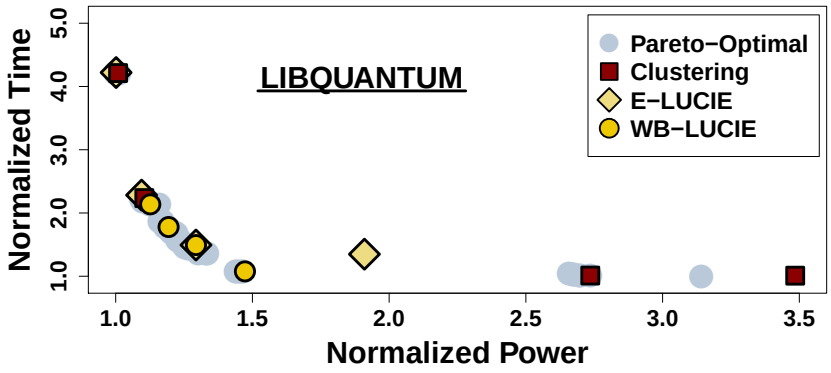


**Figure 8.17:** *Cores selected for the* ***SPECint 2006*** *suite by the Clustering, E-LUCIE, and WB-LUCIE strategies, shown for the* libquantum *benchmark.*

*libquantum*, the Clustering and E-LUCIE selections behave similarly. There are two reasons that WB-LUCIE has such good SO for *libquantum*. First, its fastest core is almost as fast as the fastest cores in the Clustering selection while consuming far less power. Second, its slowest core is noticeably faster than the slowest cores in the Clustering and E-LUCIE selections.

**Figure 8.18** shows SO for the EEMBC suite. The differences in the selection algorithms are less pronounced than for SPEC owing to the larger number of benchmarks that must be satisfied by the cores. The Clustering selection is only 1% worse than E-LUCIE. While the best selection strategy for SPEC is WB-LUCIE, for EEMBC, BE-LUCIE is, on average, 3pp (percentage points) better. This is caused by a side effect of the move counter in WB-LUCIE. The highest-performance core selected by

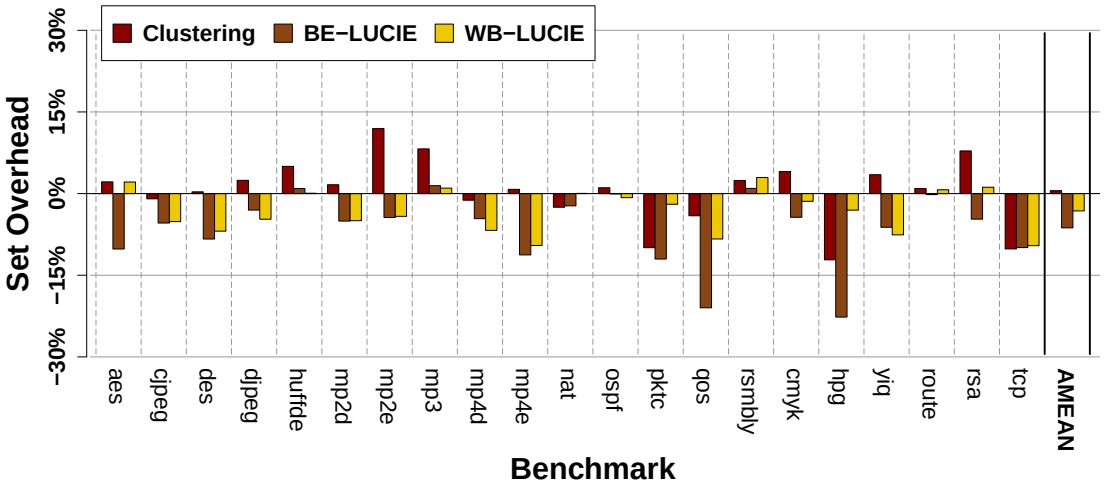


**Figure 8.18:** *Set overhead relative to the E-LUCIE selection for the **EEMBC** suite. Average SO values are 1%, -6%, and -3%, respectively. Smaller values are better.*

BE-LUCIE is the fastest core in the knee of the Pareto-optimal frontier, and is nearly as fast as the cores in the high-power, low-affinity cluster. In the case of WB-LUCIE, this core is surrounded on either side by cores with large move counters, and is removed early. (See **figure 7.3** (p. 129) and **figure 7.4** (p. 131).) Since only four cores can be selected, WB-LUCIE eventually removes all cores in the low-affinity cluster, and is left with a slightly slower fast core than BE-LUCIE.

As noted in **section 6.8.3.3** (p. 97), set overhead is evaluated over the common power range of the given two selections. For most benchmarks, the SO metric shows the WB-LUCIE selections to be faster than the E-LUCIE selections. However, the E-LUCIE selections have marginally higher availabilities than WB-LUCIE selections (see **figure 8.10** and **figure 8.11**). This raises the issue of whether one should prefer a slower selection with higher availability, or a faster selection with lower availability. In some cases, the differences will be so small as to be inconsequential, and in other cases, the processor's design goals will provide the answer. There can, however, be cases where the differences are significant, and there is no prior guidance as to which option a designer should prefer. The effective speed metric in the next section can help clarify the problem.

## 8.9 Effective Speed

Effective speed, ES, evaluates the average throughput of a set of cores under a probabilistically varying power budget. It is defined in **section 6.10** (p. 103). ES incorporates

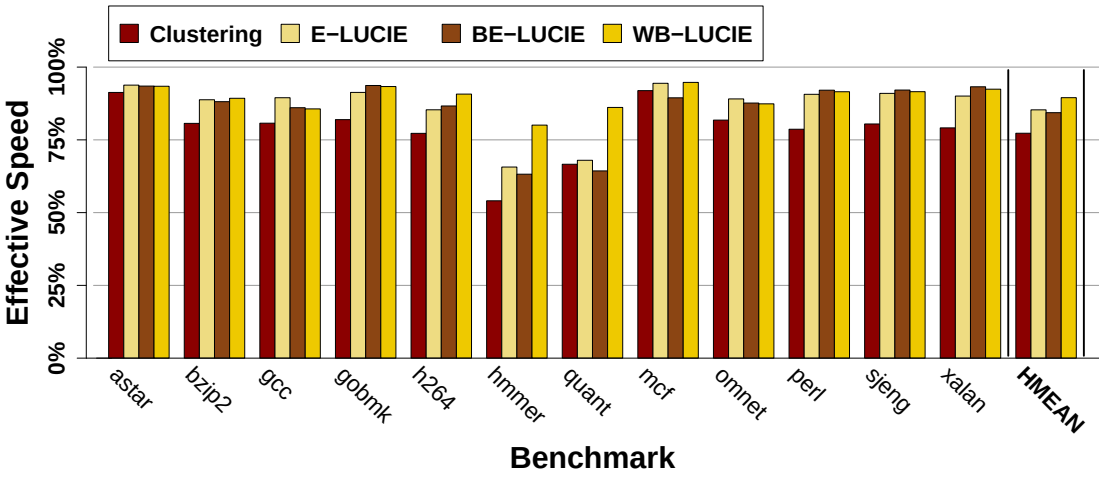


**Figure 8.19:** *Effective speed for the **SPECint 2006** suite using a high maximum power. Average ES values are 77%, 85%, 84%, and 89%, respectively. Larger values are better.*

availability and measures the overall throughput of a selection, but as a consequence of using availability, ES hides the amount of time that a task is stalled due to a lack of power. ES is evaluated over the range of power values from the power of the lowest-power benchmark-core combination in the design space to the power of the highest-power benchmark-core combination of any of the four selections. The power PDF is assumed to be flat.

**Figure 8.19** shows the effective speed of the SPEC benchmarks on each selection of cores. WB-LUCIE has the best average performance across all benchmarks, with an ES of 89%. Compared to the other versions of LUCIE, the move counter in WB-LUCIE leads to the inclusion of a slightly faster core that improves average speed. Average ES for the Clustering strategy is 77%. The greatest difference between ES for the Clustering selection and the WB-LUCIE selection is for the *hmmer* benchmark. WB-LUCIE provides an ES of 80%; the ES for the Clustering strategy is only 54%. **Figure 8.20** plots the two selections for *hmmer*. There is nothing unusual about *hmmer* that is causing the disparity in ES. *hmmer* simply has a very large operating range along both the power and time axes (as can also be seen in **figure 8.6** and **figure 8.7**). The result is a substantial slowdown for the Clustering algorithm when there is not enough power to run on the two fastest cores. WB-LUCIE performs much better due to the concentration of cores around the knee of the Pareto-optimal frontier.

The analysis in **figure 8.19** is quite generous to the Clustering strategy, since the Clustering strategy selects the highest-power core and therefore sets the maximum

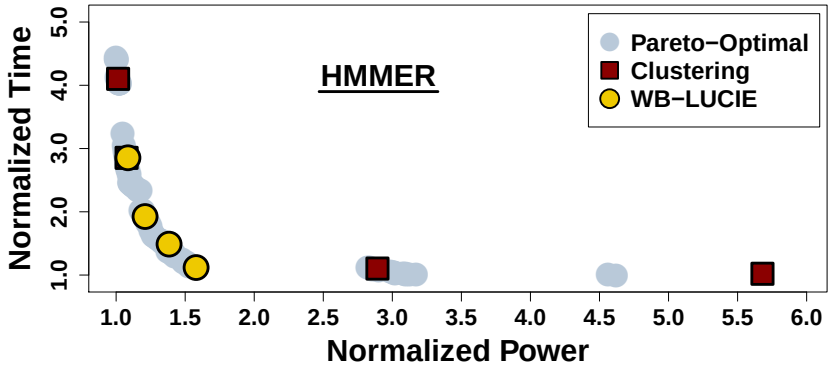


**Figure 8.20:** *Cores selected for the **SPECint 2006** suite by the Clustering and WB-LUCIE strategies, shown for the* hmmer *benchmark. The second core in the Clustering selection is hidden behind the lowest-power WB-LUCIE core.*

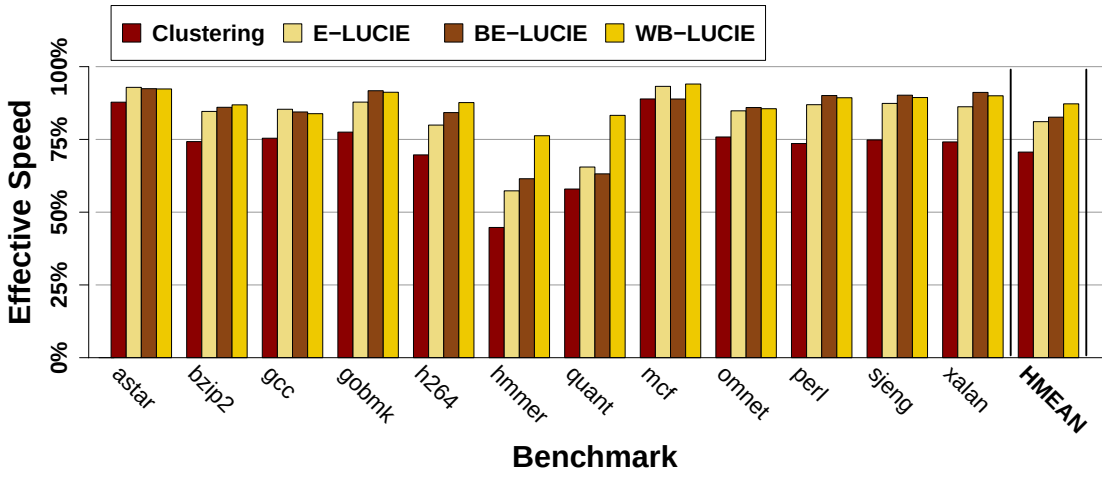


**Figure 8.21:** *Effective speed for the **SPECint 2006** suite using a low maximum power. Average ES values are 71%, 81%, 83%, and 87%, respectively. Larger values are better.*

power available to the processor. This is somewhat unfair to the LUCIE strategies, which select cores that consume far less power. **Figure 8.21** shows effective speed for the SPEC suite when maximum available power is based on the highest-power benchmark-core combination using only LUCIE selections. ES for WB-LUCIE is reduced by only two percentage points (2pp) from 89% to 87%, while ES for the Clustering selection is reduced by 6pp to 71%. This again demonstrates the effectiveness of the LUCIE algorithm for selecting cores for power-constrained devices.

**Figure 8.22** shows effective speed for the EEMBC suite. The results for EEMBC are very similar to SPEC results. The significant difference is that for the EEMBC suite, BE-LUCIE performs marginally better than WB-LUCIE—average ES is 2pp

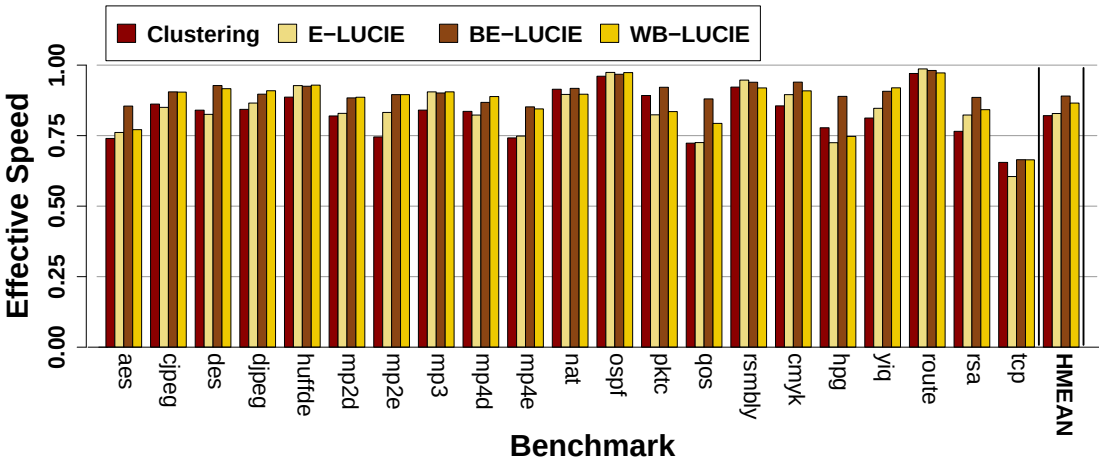


**Figure 8.22:** *Effective speed for the **EEMBC** suite using a high maximum power. Average ES values are 82%, 83%, 89%, and 87%, respectively. Larger values are better.*

greater for BE-LUCIE. This is due to the same move counter side effect described above in **section 8.8**.

By incorporating availability and performance, effective speed helps clarify the problem of choosing between a selection with better performance but lower availability, and a selection with worse performance but higher availability (see **section 8.8**). For the SPEC suite, WB-LUCIE provides greater throughput, but for the EEMBC suite, BE-LUCIE provides greater throughput. It should, however, be noted that due to the considerations described in **section 6.10.3** (p. 105), ES cannot be used as a replacement for Av and SO.

## 8.10 Scalability

The preceding sections have considered selections of four heterogeneous cores. This section evaluates the scalability of the selection algorithms—how the algorithms behave as the number of selected cores increases. While current technology and design methodologies allow for only a small number of heterogeneous cores on a processor, the number will inevitably increase as technology progresses. Scalability is analyzed using gap overhead, monotonicity, and generality, three metrics that evaluate how selection algorithms respond to increasing core counts.

### 8.10.1 Gap Overhead

Gap overhead, GO, evaluates how close the performance of a selection of cores is to an ideal selection containing all cores in the candidate set. It is defined in **section 6.7** (p. 88). To ensure that the GO evaluation is comparable across different selections, the core in the candidate set with the minimum average power (core #988) is pinned. Maximum power is set to be the power of the highest-power benchmark-core combination in any selection of 16 cores for any version of LUCIE. The power PDF is assumed to be flat. The Clustering strategy is excluded form this comparison, as it does not support core pinning.

**Figure 8.23** shows the minimum, maximum, and average GO for the SPEC benchmark suite as the number of selected cores increases from two to 16. At two cores, average GO for E-LUCIE is 23%. At four cores, this has dropped to 6%, and at eight cores, to 3%. BE-LUCIE and WB-LUCIE start with lower average GO values—10% and 11% respectively. However, at four cores, both have a GO of 8%, and at eight cores, GO is 3% and 4%, respectively. E-LUCIE starts out with a high GO value that gradually decreases and converges at a lower value than GO for either BE-LUCIE or WB-LUCIE. The rate of GO decrease for BE-LUCIE and WB-LUCIE is also less regular than the decrease for E-LUCIE. Up until eight cores, worst-case GO is better for BE-LUCIE and WB-LUCIE than for E-LUCIE.

The differences between E-LUCIE, and BE-LUCIE and WB-LUCIE can be explained by the biased distance metric used in the latter two. The biased distance metric emphasizes cores in the knee of the power-performance trade-off curve. This approach is particularly effective for approximating the candidate set with very few cores. However, as the number of cores increases, the potential for performance in the knee region is quickly exhausted. E-LUCIE selects cores with a broader spread (see **figure 8.6**), and can sometimes outperform BE-LUCIE and WB-LUCIE in terms of gap overhead. As the number of selected cores increases, all selections will eventually converge on the candidate set.

The gap overhead results in this section are not directly comparable to the earlier analyses in this chapter, since gap overhead is evaluated with a pinned core and a maximum available power based on 16 selected cores. GO results are similar for the EEMBC suite, but since there are more EEMBC benchmarks, more core types are required before GO plateaus.

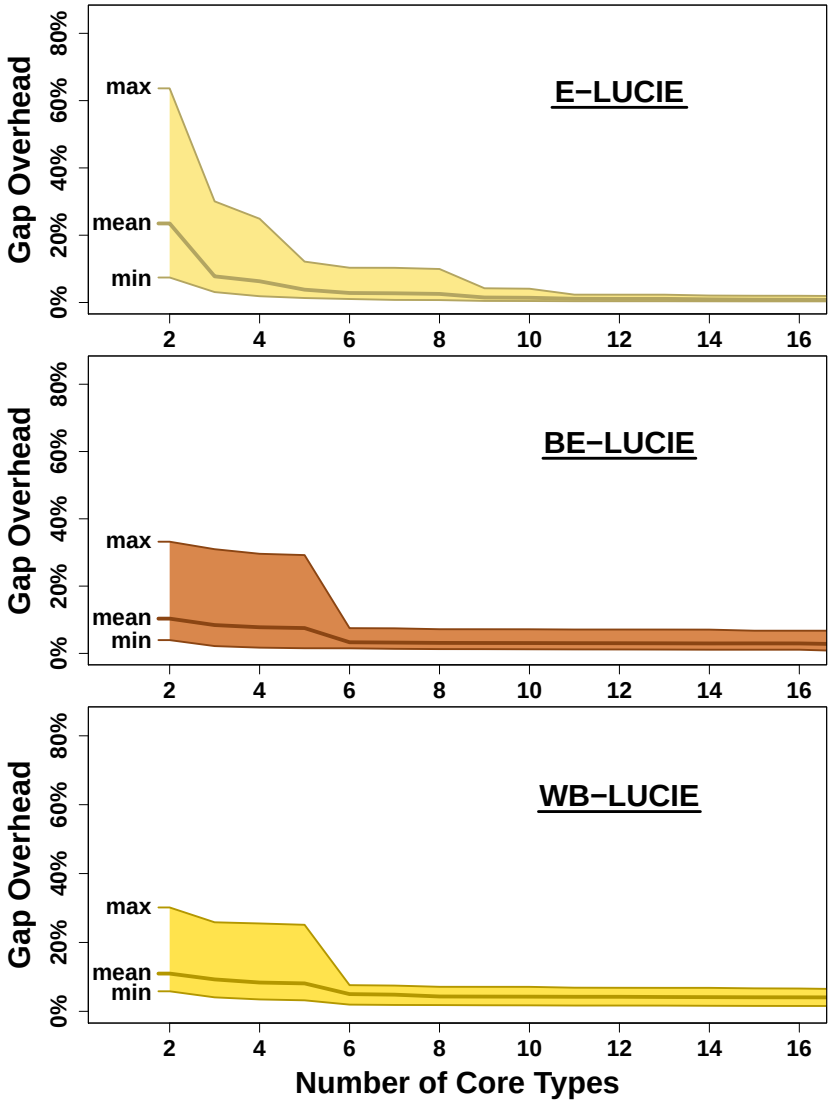


**Figure 8.23:** *Gap overhead as the three versions of the LUCIE algorithm select increasing numbers of cores for the **SPECint 2006** suite. Smaller values are better.*

### 8.10.2 Generality

The generality metric, $\mathbb{G}$, evaluates the average fraction of cores that are useful to a given benchmark. It is defined in **section 6.11** (p. 106). **Figure 8.24** plots the generality for cores selected by the three versions of LUCIE and the Clustering strategy for the SPEC suite as the number of selected cores increases from two to 16. The versions of LUCIE all have similar generality, while $\mathbb{G}$ for the Clustering strategy is marginally lower. While it is expected that $\mathbb{G}$ will decrease as the number of cores increases, and while there is nothing intrinsically wrong with a low $\mathbb{G}$ value, the fact that the Clustering strategy has a $\mathbb{G}$ value less than 1.0 at two cores illustrates

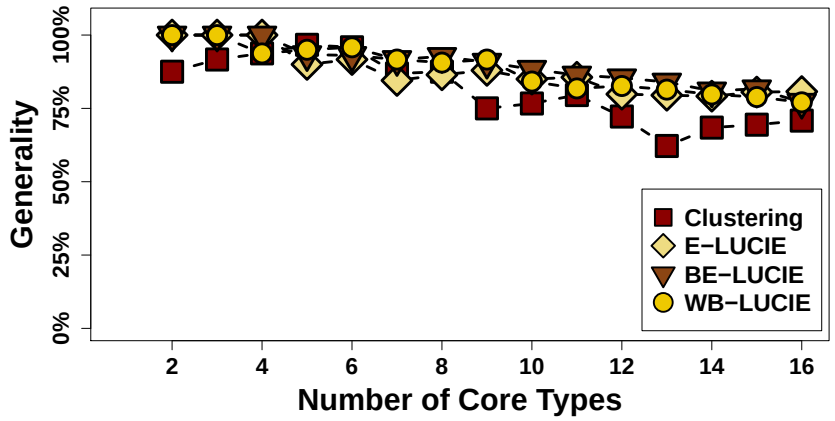


**Figure 8.24:** *Generality as the four algorithms select increasing numbers of cores for the **SPECint 2006** suite.*

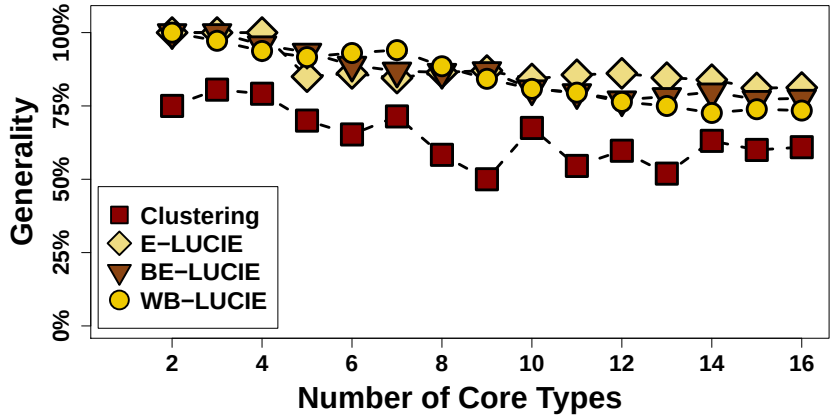


**Figure 8.25:** *Generality as the four algorithms select increasing numbers of cores for the **EEMBC** suite.*

a weakness in the strategy. It indicates that even when only two cores are selected, some benchmarks cannot use all cores—i.e., the selection is effectively homogeneous for those benchmarks. This behavior of the Clustering strategy is even more evident for the EEMBC suite, as shown in **figure 8.25**. $\mathfrak{T}$ for the Clustering strategy is consistently over 10pp (percentage points) lower than $\mathfrak{T}$ for any of the LUCIE selections. LUCIE selects cores that are more broadly applicable than cores selected by the Clustering algorithm, which is particularly important for general-purpose, consumer processors with only a few types of heterogeneous cores.

The low generality of the Clustering strategy is explained by its preference for disproportionately high-power cores, such as core #319. These cores do not provide a performance advantage to all benchmarks. They are therefore not Pareto-optimal for all benchmarks, and lower the generality of the selection. For LUCIE algorithms, as

the number of cores increases, E-LUCIE maintains the highest Ъ, and WB-LUCIE has the lowest. This is because BE-LUCIE and WB-LUCIE focus cores near the knee of the power-performance trade-off. For larger numbers of cores, the cores are in a tighter cluster, and there is a greater chance of *shadowing*, where for a given benchmark, a core will have both better power and better performance than another core. If a designer is implementing a large number of cores, then this per-benchmark specialization may be a desirable feature.

### 8.10.3 Monotonicity

The monotonicity metric, ר, evaluates how dependent the ordering of cores is on the task being executed. It is defined in **section 6.12** (p. 110). **Figure 8.26** plots the monotonicity for cores selected by the three versions of LUCIE and the Clustering strategy for the SPEC suite as the number of selected cores increases from two to 16. The Clustering strategy has a consistently high ר relative to the LUCIE strategies. BE-LUCIE has a particularly low ר. While a low monotonicity is not intrinsically problematic, it indicates that scheduling can be difficult.

The high monotonicity of the Clustering algorithm is the result of its strategy of maximizing consistency—cores that have more consistent behaviors across benchmarks tend to also have more consistent orderings across benchmarks. LUCIE does not consider consistency, which makes it more likely that two similar cores are selected if that benefits the benchmarks. When cores are similar, per-benchmark behavior variations can easily cause the order of the cores to switch. BE-LUCIE and WB-LUCIE have lower monotonicities than E-LUCIE for the same reason that they have lower generalities. The two strategies focus cores on the knee of the power-performance trade-off, and since the cores are closer together, there is a greater chance that their orderings will change.

**Figure 8.27** shows monotonicity for the EEMBC suite. The monotonicities are even more varied for BE-LUCIE and WB-LUCIE than for SPEC, but the underlying reason is the same: These two variants of LUCIE focus cores in the knee of the power-performance trade-off, and given the 21 benchmarks in the EEMBC suite, there is a high chance of core orderings changing.

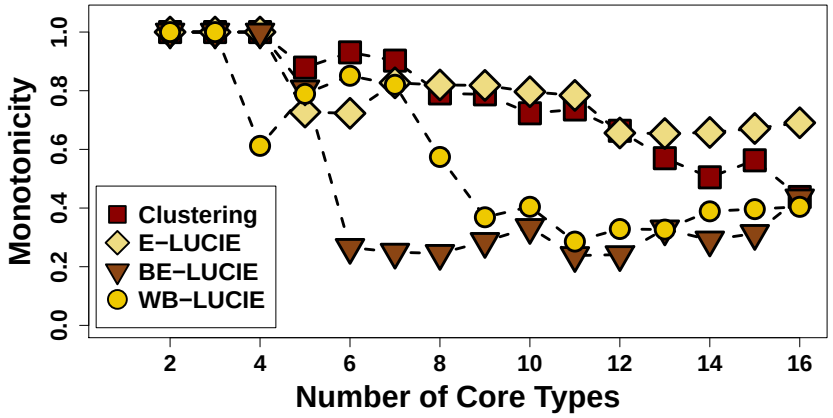


**Figure 8.26:** *Monotonicity as the four algorithms select increasing numbers of cores for the **SPECint 2006** suite.*

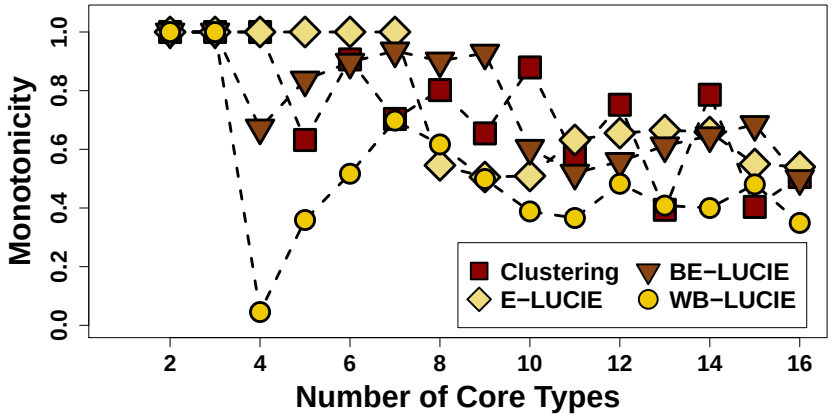


**Figure 8.27:** *Monotonicity as the four algorithms select increasing numbers of cores for the **EEMBC** suite.*

## 8.11 Selection with a PDF

**Section 7.9** (p. 134) described how a power probability density function can be used to direct LUCIE to select cores from more important regions of the design space. **Section 6.13** (p. 113) described using the same PDF to weight metrics and evaluate selections of cores that target a weighted space. This section uses set overhead to evaluate cores selected for the EEMBC suite using the PDF shown in **figure 7.8** (p. 136). **Figure 8.28** graphs the results of selecting four cores with the Clustering strategy, four cores with WB-LUCIE, and four cores with WB-LUCIE that is guided by the PDF. Cores selected by WB-LUCIE (with PDF) are listed in **table 8.6**. SO is calculated using the same PDF. Maximum power is the power of the highest-power benchmark-core combination of any of the four selections. The baseline selection is the set of four

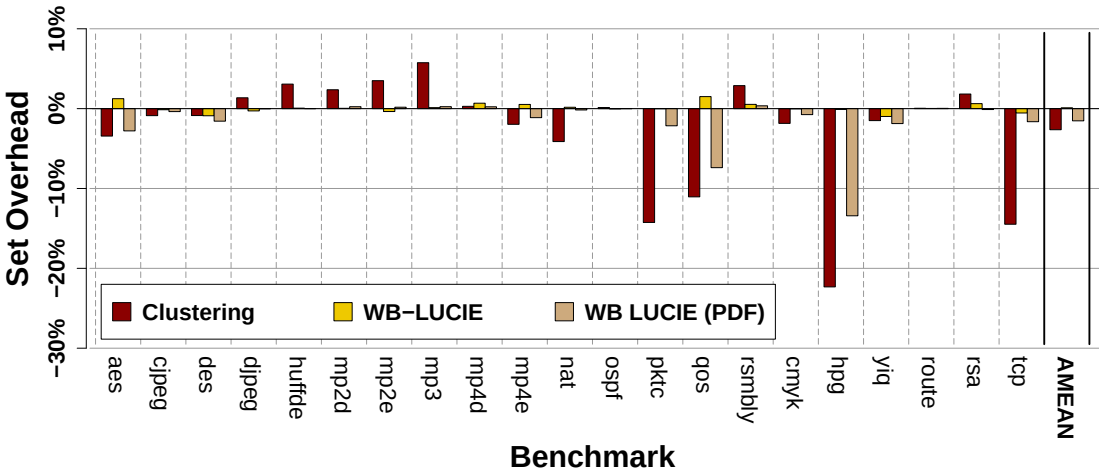


**Figure 8.28:** *Set overhead relative to the E-LUCIE selection for the **EEMBC** suite. SO is improved when WB-LUCIE has access to the power PDF. Average SO values are -3%, 0%, and -2%, respectively. Smaller values are better.*

| EEMBC Cores | | | | |
|---|---|---|---|---|
| **WB-LUCIE Selection (PDF)** | #1056 | #821 | #424 | #671 |

**Table 8.6:** *Four cores selected by WB-LUCIE for the **EEMBC** benchmark suite given the power PDF in **figure 7.8** (p. 136). Cores are ordered from low to high power. Core configurations are in **appendix A**.*

cores selected by E-LUCIE without access to the PDF—i.e., the same baseline as in **section 8.8**.

For this particular PDF, WB-LUCIE cores are effectively identical to E-LUCIE cores. The PDF has a large high-power component, and as a result, the Clustering strategy with its extreme cores substantially improves performance for some benchmarks. Most notably, *rgbhpgv2* is 22% faster than on E-LUCIE. Improvements from WB-LUCIE (PDF) are smaller, but more consistent. On average, Clustering and WB-LUCIE (PDF) selection improve performance by 2% and 3%, respectively.

There is no mechanism to communicate a power PDF to the Clustering strategy. The Clustering strategy happens to perform well for this particular PDF, but the result is not guaranteed for all PDFs In contrast, LUCIE is guided by the PDF, so the improvements in this example are transferable to other PDFs.

## 8.12 Conclusion: The Best Algorithm

This evaluation chapter has had two purposes: to demonstrate the usage of the metrics defined in **chapter 6**, and to evaluate the LUCIE algorithm defined in **chapter 7**. The demonstration aspect of the chapter is obvious—each metric can be used to gain new insight into a selection of cores. The question of identifying the "best" selection algorithm is not immediately clear, however. The GA strategy can be discarded due to the low availability of its selections. The cores are not appropriate for mobile devices, though there may be other devices for which the GA strategy works well. The Clustering strategy produces sets of cores with surprisingly good characteristics, given that the algorithm is designed for use with server processors. Nevertheless, cores selected by the Clustering strategy provide consistently slower speed than cores selected by LUCIE, as can be seen from the set overhead and effective speed evaluations. The Clustering strategy is also unwieldy and lacks flexibility—it is non-deterministic, and there is no facility to either pin cores or to provide a power PDF.

None of the three versions of the LUCIE algorithm are consistently superior to the other two. For example, for the EEMBC suite, set overhead shows that a biased distance metric without a move counter (BE-LUCIE) leads to the best results. In contrast, for the SPEC suite, biased distance hurts performance unless it is coupled with a move counter (WB-LUCIE). It is possible to create an endless series of variations on the LUCIE algorithm with different types of distance metrics, different handling of the move counter, and other modifying features. The consistency of the LUCIE results, and the fact that LUCIE is consistently superior to the current state-of-the-art GA and Clustering strategies, demonstrates that the underlying iterative core elimination mechanism is sound. In some cases, the processor designer will have insight into the behavior of the representative suite of benchmarks, and will be able to use the most appropriate version of LUCIE. Barring any further insights, WB-LUCIE is the recommended version of the algorithm. It focuses cores to the knee of the power-performance trade-off, thereby maximizing flexibility, while using a move counter to avoid too many cores in the knee.

## 8.13 Summary

Using both qualitative and quantitative evaluations, this chapter has shown that current state-of-the-art core selection algorithms either cannot select cores for diversity, or

cannot take advantage of variations in task behavior. In contrast, the LUCIE algorithm produces highly available selections of cores that maximize performance under strict power limits. By evaluating cores based on their contributions to individual tasks, LUCIE is able to select a small number of generally applicable cores, or a larger number of more specialized cores.

# Chapter 9:

# Conclusions

Heterogeneous processor design is dependent on methods of evaluation and methods of core selection. **Chapter 5** motivated the evaluation and selection problems for single-ISA heterogeneous processors, and established a theoretical framework for reasoning about the problems. **Chapter 6** defined a series of metrics for evaluating sets of heterogeneous cores under a probabilistically varying power budget. **Chapter 7** defined the *LUCIE* algorithm for core selection. **Chapter 8** demonstrated the metrics by comparing LUCIE to state-of-the-art selection algorithms. The contributions of these technical chapters are summarized in greater detail below. This is followed by a critical analysis of the contributions, and an introduction of potential avenues for further research.

## 9.1 Contributions

This thesis has made contributions in three general areas: The first is the definition of a theoretical framework and set of assumptions required for addressing the evaluation and design problems. The second is a set of metrics for evaluating selections of heterogeneous cores. The third is an algorithm for selecting cores from a candidate set. The contributions in these areas are summarized in turn.

### 9.1.1 Motivating Framework

**Chapter 5** contributed a detailed problem description and a runtime model. These two form a framework for solving the evaluation and selection problems. It was argued that the crucial feature of a processor intended for a power-constrained device is a set of cores the enable the flexibility to run tasks under various power budgets. In a heterogeneous processor, runtime flexibility is provided by a diverse selection of cores.

It was further argued that current evaluation techniques cannot adequately measure diversity or performance under a varying power budget, and that current selection techniques do not adequately select for diversity. To address the evaluation and selection problems, **chapter 5** defined a set of assumptions about runtime behavior—the runtime model. The runtime model represents the combined behavior of an entire device with an available power probability density function (PDF). By evaluating selections of cores with respect to a power PDF, it is possible to analyze even very complex devices. The evaluation and selection methods in later chapters are based on a power PDF.

### 9.1.2 Evaluation Metrics

**Chapter 6** defined eight new metrics for evaluating selections of heterogeneous cores. The metrics assume a mobile device where performance must be maximized under a strict but variable power budget. The uniformity of a selection can be measured with the *KS test* and the *localized non-uniformity* metric. *Gap overhead* measures slowdown over an ideal selection of cores. *Set overhead* measures slowdown over a different selection of cores. *Availability* measures the likelihood that a selection is available to do work. *Effective speed* measures the average speed of a selection. *Generality* measures the extent to which cores are specialized to specific benchmarks, and *monotonicity* measures how dependent the order of cores is on the benchmark.

The metrics comprise a set of tools that can be used to analyze various features of selections of cores. One, some, or all metrics can be used, as dictated by the requirements of the processor. The metrics are oblivious to the implementation details of cores, and are only based on the power and performance of cores. This ensures that the metrics are portable to different design spaces and use cases.

### 9.1.3 Core Selection

**Chapter 7** defined three variations of the LUCIE algorithm for selecting cores, and **chapter 8** compared LUCIE to state-of-the-art selection methods. LUCIE uses the set of power-performance Pareto-optimal cores in a design space as the standard for maximum diversity, and approximates the optimal set using a small set of cores. The LUCIE algorithm is unique in that it optimizes for the runtime flexibility seen by individual benchmarks. As a consequence, it allows cores to fulfill different roles for different benchmarks. LUCIE addresses the requirements of power-limited devices by selecting diverse cores but avoiding extreme cores. LUCIE supports core pinning, can

optionally use an available power PDF to direct selection, and unlike existing selection methods, is deterministic.

## 9.2 Critical Analysis

The general multicore processor design problem described in **section 1.1** (p. 1) is intractable with current techniques. By limiting the problem to only selecting core types and to only a single ISA, this thesis has been able to suggest solutions to both the evaluation and design aspects of the problem. The following sections present a critical analysis of the proposed theoretical framework, metrics, and selection algorithm.

### 9.2.1 Motivating Framework

The purpose of the motivating framework is to reduce the general processor design problem to a manageable size. Reducing the design problem to CPU cores of only one ISA is a somewhat limiting, but justifiable decision. There are no compelling reasons for including cores that implement different, general-purpose CPU ISAs on a single processor. The benefits of including GPUs, DSPs, and other accelerators have been studied extensively. However, the use of specialized hardware does not negate the need to understand the benefits of microarchitectural heterogeneity in CPU cores.

Reducing the design problem to selecting cores rather than designing entire processors is necessary for understanding the limits of the benefits that single-ISA heterogeneity can deliver. It must be possible for a designer to determine the best set of heterogeneous cores to meet a processor's design requirements. However, it is conceivable that once uncore components are factored in, the best set of cores will be practically indistinguishable from another heterogeneous set. For example, the best set might contain four different cores, but once memory bottlenecks are introduced, the behavior of the set of four could be indistinguishable from the behavior of a set with only three core types. Analysis of the selection of cores will still be valuable, because it helps the designer find bottlenecks in the processor. In some cases, though, it might not be possible to remove the bottlenecks, and the full benefits of heterogeneity as predicted from the set of cores might not be realizable.

The example dataset further reduces the design problem to only out-of-order cores. This reduction is due to limitations in the gem5 and McPAT models, rather than a need to manage the design and evaluation problems. The metrics and the LUCIE algorithm

can be applied without modification to a design space that contains in-order and out-of-order cores or even just in-order cores.

The runtime model in **section 5.4** (p. 58) is based on several assumptions. The most significant of these is the oracle scheduler. The scheduler is assumed to have prior knowledge about the power consumption of all tasks on all cores, and is assumed to be able to fairly partition power among tasks. A processor in the near future could well have the ability to monitor each core's power consumption, which would enable a scheduler to learn the possible power-performance trade-offs for tasks. However, the power and performance of a task on a core can also be dependent on input from the user, and this is much more difficult to predict. Similarly to uncore components, prediction error and imperfect fairness in the scheduler may mask some of the benefits of heterogeneity

### 9.2.2 Evaluation Metrics

Five aspects of the evaluation metrics benefit from further attention. These are described in turn.

**Multiple ISAs.** While the example dataset and runtime model do not explicitly consider multiple ISAs or accelerators, most of the metrics can be easily extended beyond the single-ISA case. The eight metrics in **chapter 6** are all based on the power and performance of cores and not on implementation details—i.e., the metrics are oblivious to the ISA. If, for example, a task can run on a GPU as well as on the CPU, then the power and performance of the task on the GPU need only be added to the set of possible power-performance points, and the metrics can be evaluated normally. The exceptions are the uniformity metrics. While CPU cores can be expected to be roughly uniformly distributed in the power-performance space, the gap between CPU cores and a GPU is likely to be large. The KS test will be dominated by this gap, and will not provide useful information about the selection of CPU cores. The localized non-uniformity metric continues to be sensitive to small gaps in the presence of large gaps, but when a large gap is inevitable, greater care must be taken in interpreting the $\supset$ value. Similar considerations apply to processors that have instruction set extensions, DSPs or other specialized accelerators, or even multiple CPU ISAs.

**DVFS.** The example dataset excludes DVFS. Since DVFS is simply a way of adjusting a core's power and performance, the metrics can handle DVFS in the same way as they handle accelerators. I.e., the metrics can be used to evaluate all Pareto-optimal

power-performance points for a task, regardless of whether the points are provided by different DVFS levels applied to the same CPU core, different CPU cores, or specialized hardware. Care must be taken when analyzing the localized non-uniformity, generality, and monotonicity of sets of cores with DVFS. When cores are not uniformly spaced, not general, or not monotonic, then these are significant features of which the designer must be aware. It is a far less significant issue when high non-uniformity, low generality, or low monotonicity are caused by DVFS levels. It may be better to evaluate these three metrics using, for example, only the lowest and highest DVFS levels of each core.

**Contention.** The metrics assume that there is no contention for cores. That is, it is assumed that when there is enough power to use a core, then the core can be used. A processor in the dark silicon regime can have many more cores than can be powered at a time, so this assumption is not unrealistic. However, any real scheduler must inevitably handle contention. There will be times when there is enough power for a core, but the core will not be available for work because it is busy. This has a complicating effect on metrics. The metrics, as defined, assume a fixed set of cores and a variable power budget. If core contention is added to the runtime model, then the metrics must handle both a probabilistic power budget and a probabilistic set of cores. While this is possible, it adds both an extra dimension of complexity to the metrics and a another potential source of errors.

**Available power range.** The available power assumptions of the runtime model can have large, non-obvious effects on the gap overhead, set overhead, availability, and effective speed metrics. For example, if the maximum available power is increased while the selection of cores is unchanged, then availability increases. This is because the fraction of the power PDF above the power of the lowest-power core increases. The dependence on the available power range is not a weakness of the metrics—the metrics function precisely as intended. However, care must be taken to ensure that the power PDF is defined correctly.

**Ideal performance estimate.** The gap and set overhead metrics assume that the designer has access to $\overline{Y_i}$, an estimate of the best possible performance in a power range. In some circumstances, it may be difficult to determine $\overline{Y_i}$. In these cases, GO and SO will continue to be accurate in relative terms, but can be wrong in absolute terms. For example, if $\overline{Y_i}$ is optimistic, then GO will be pessimistic, and will overestimate how far the selection is from the ideal.

### 9.2.3 Core Selection

The LUCIE selection algorithm departs from the common processor design approach of using a linear cost model. A cost model allows the configurations of various components to be traded off to arrive at an optimal processor. Some of the problems with these optimization techniques are described in **section 2.2.3** (p. 10), **section 3.3.1** (p. 28), and **section 5.6.2** (p. 64). Fundamentally, the cost models assume that it is possible to trade off various metrics using continuous functions. This assumption is highly suspect. It is often assumed, for example, that delay is twice as valuable as energy. However, if a real processor's delay reaches a certain level, the processor will be commercially unviable regardless of how low its energy consumption. Similarly, microarchitectural structures can only be set to discrete sizes. Data cache size serves as an example (see **section 4.3.3** (p. 44)). Increasing data cache size by one increment from 32kB to 64kB results in a substantial increase in power consumption. A few benchmarks receive performance benefits from the increase; most do not. Only one of the two sizes can be chosen—there is no way to interpolate a compromise size between a low-power, 32kB cache and a high-power, 64kB cache to provide some performance benefit to benchmarks that can use the larger cache without increasing the other benchmarks' power consumptions too much. In short, linear models assume a continuous design space that does not correspond to reality.

Linear models do have their advantages, in that they can trade off disparate quantities. A linear cost model could help determine, for example, whether to add a DVFS level to a core or to add an additional core type to a processor, whether to use a large amount of silicon area for a single GPU or to use the same area for several CPU cores, and whether to add another instance of an existing core or a new core type. The LUCIE selection algorithm does not use a standard cost model, and is therefore incapable of making these types of trade-offs. Instead, LUCIE selects power-performance points and assumes that cores implementing those points are interchangeable in terms of cost—implementing one power-performance point instead of another will not have a significant effect on silicon area, engineering effort, manufacturing costs, etc. While this assumption is approximately correct when the cores in the candidate set are similar, it becomes increasingly tenuous as more exotic options are included. Implementing some power-performance points with DVFS is relatively easy and cheap; implementing power-performance points with specialized accelerators could be much more involved. This leaves the processor designer with a

difficult choice. On the one hand, he could use LUCIE to select cores, but LUCIE is not aware of the relative cost differences of more interesting, exotic components. On the other hand, he could devise a linear model that can optimize all manner of components. However, as already noted, such a model cannot capture a real design space. A further issue is that even if a processor's design space were linear, it seems unlikely that the true trade-offs between components could be described accurately and consistently. For example, determining whether a processor should contain an additional core or an additional accelerator is dependent on the respective speed, power, and area of the core and accelerator; how frequently the accelerator can be used; how many other cores there are of the same type as the given core; how many cores of other types there are; etc. It is unclear how functions to describe these relationships could be defined with enough precision to make accurate design decisions. It may be possible to extend LUCIE to better handle more exotic scenarios, but it is most likely impossible to shoehorn a discrete processor design space into a linear model.

Linear models and other core selection methods optimize a single metric. For example, the GA selection strategy optimizes speed, and the Clustering selection strategy optimizes consistency. LUCIE does not optimize a single metric. Instead, it attempts to evaluate the relative significance of cores to the power-performance trade-off, and to select the most significant ones. This can make LUCIE somewhat difficult to analyze, since there is no single figure of merit against which it can be evaluated. For example, **chapter 8** showed that LUCIE is clearly superior to competing selection algorithms. It is less clear which of the three versions of LUCIE is best. Each version has a slightly different definition of significance, and it is not obvious whether one definition is always better than the others. This is ultimately because there is no single, universal figure of merit that a processor must unqualifiedly optimize. Without such a "gold standard," there will always be some ambiguity in evaluating selections.

## 9.3 Future Work

There are several possible avenues for extending the material presented in this thesis. These include runtime analysis, trade-off analysis, further work in selection, and further work in evaluation.

**Runtime analysis.** The runtime model and metrics provide a unique opportunity for studying quality-of-service (QoS) and user satisfaction issues. If a task frequently switches between running quickly and slowly, it could have acceptable average perfor-

mance, but a user would be dissatisfied with the slow phases and uneven performance. This could happen if, for example, a game is repeatedly switched between a power-hungry core when the processor is cool, and a low-power core when the processor heats up. The runtime model guarantees constant performance for the duration of a scheduling interval, so an empirically determined available power PDF can be used to calculate the potential variation in behavior across scheduling intervals. The scheduler could then be tuned to minimize the amount of time that the task spends on a core that is faster than required, so that heat and power consumption are reduced, and the amount of time that the task must spend on a core that is too slow is also minimized. Using the runtime model, a set of available power PDFs for various tasks, and metrics like set overhead, a designer can optimize a selection of cores and a scheduler to provide adequate QoS in various usage scenarios. A QoS study could further be extended to explore how close realistic schedulers can come to the oracle scheduler assumed in this thesis.

**Trade-off Analysis.** While all the discussions and examples in this thesis have used a power-performance trade-off, this is not the only trade-off possible. The LUCIE algorithm and many of the metrics could be used with any pair of quantities that are inversely correlated. A particularly interesting option is a power-energy trade-off. Many authors have observed that computational sprinting is preferable to pacing—energy is minimized when tasks are executed quickly at high power, since cores can then enter low-power states sooner (see **section 3.2.2** (p. 19)). This observation has consequences for mobile processor design. Mobile devices must minimize power due to thermal considerations, and they must minimize energy consumption due to battery considerations. A power-energy trade-off suggests that tasks should be run as quickly as possible within thermal limits to maximize battery life. The exception is tasks where the execution time is not directly dependent on the performance of a core. For example, a video will take the same amount of time to watch regardless of whether a low-complexity or a high-complexity version is viewed. The low-complexity version is likely to consume less power, and since the execution time is constant, less energy.

**Selection.** The LUCIE algorithm treats each core-benchmark combination as a single point in the power-performance space, and operates only on cores that are Pareto-optimal for at least one benchmark. However, it is possible to adjust a core's power and performance using DVFS. It is also conceivable that in some cases, a good core is not Pareto-optimal for any benchmark, but is a compromise between several optimal cores. Future work could extend LUCIE to treat each core as a set of power-

performance points, where each point corresponds to a DVFS level. The distance metric could then be adjusted to evaluate a core based on, for example, how far its furthest DVFS level is from any other core's DVFS levels. LUCIE could also be seeded with both Pareto-optimal cores and nearly Pareto-optimal cores from the design space exploration stage. This approach may result in slightly better, compromise cores, though LUCIE would require modifications so that it does not immediately eliminate cores with an affinity of 0.0.

**Evaluation.** Finally, there is scope for further research on evaluation techniques, particularly on metrics for uncore components and entire processors. An optimistic goal is the universal metric noted in the previous section—a metric that all processors must maximize. If such a metric were to exist, then all processors and all processor design techniques could be compared unambiguously. Based on the range of metrics discussed in the literature as well as the contributions of this thesis, the existence of a grand, unified metric appears highly improbable. In the absence of a universal metric, the best a designer can do is carefully consider design methodologies and carefully weigh a range of metrics. Metrics and methodologies that oversimplify the design problem will mask important details, and metrics and methodologies that reveal too many details can paralyze the design process with information overload.

# Appendix A:

# Design Space

**Table A.1** lists the configurations of all 3000 cores in the example dataset. The example dataset is described in **section 4.3** (p. 41). The set of values that each microarchitectural parameter can take is listed in **table 4.3** (p. 43). The dataset contains simulated power and performance data for each core when running each of 33 benchmarks. For cores that are referenced in the body of the thesis, the first reference is listed in the "Page" column.

**Table A.1:** *List of all cores in the example dataset.*

| | Caches | | | | Registers | | Queues | | | | Branch Predictor | | | | | | | BTB | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Core** | **DS** | **DW** | **IS** | **IW** | **IR** | **FPR** | **IQ** | **LQ** | **SQ** | **ROB** | **GCB** | **GE** | **LCB** | **LHB** | **LHE** | **CCB** | **CE** | **BE** | **BT** | **Page** |
| **#1** | 64kB | 2 | 4kB | 4 | 128 | 96 | 32 | 32 | 8 | 64 | 3 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2** | 32kB | 2 | 64kB | 2 | 96 | 128 | 64 | 64 | 8 | 128 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#3** | 64kB | 2 | 4kB | 4 | 256 | 128 | 32 | 32 | 16 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#4** | 64kB | 1 | 64kB | 4 | 64 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#5** | 16kB | 1 | 64kB | 1 | 96 | 256 | 32 | 8 | 8 | 64 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#6** | 64kB | 4 | 4kB | 1 | 96 | 256 | 32 | 64 | 32 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#7** | 64kB | 1 | 64kB | 2 | 96 | 96 | 32 | 64 | 16 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#8** | 32kB | 4 | 32kB | 2 | 96 | 128 | 64 | 16 | 32 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#9** | 16kB | 2 | 32kB | 4 | 256 | 256 | 32 | 16 | 8 | 16 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#10** | 32kB | 1 | 32kB | 1 | 128 | 256 | 64 | 8 | 8 | 16 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#11** | 16kB | 2 | 4kB | 1 | 96 | 128 | 32 | 16 | 64 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#12** | 64kB | 1 | 32kB | 2 | 96 | 256 | 32 | 16 | 32 | 64 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#13** | 32kB | 2 | 32kB | 2 | 50 | 128 | 64 | 16 | 16 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#14** | 64kB | 1 | 4kB | 4 | 256 | 256 | 64 | 16 | 8 | 32 | 1 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#15** | 16kB | 1 | 32kB | 1 | 50 | 256 | 32 | 16 | 8 | 64 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#16** | 16kB | 4 | 64kB | 4 | 64 | 128 | 32 | 8 | 64 | 128 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#17** | 32kB | 4 | 16kB | 2 | 64 | 128 | 16 | 32 | 64 | 40 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#18** | 32kB | 2 | 16kB | 2 | 64 | 96 | 32 | 32 | 16 | 40 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#19** | 32kB | 4 | 4kB | 4 | 256 | 96 | 64 | 32 | 16 | 16 | 1 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#20** | 64kB | 1 | 16kB | 4 | 256 | 96 | 16 | 32 | 8 | 40 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#21** | 32kB | 1 | 16kB | 2 | 50 | 256 | 32 | 64 | 64 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#22** | 16kB | 4 | 8kB | 2 | 128 | 96 | 32 | 8 | 64 | 16 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#23** | 16kB | 4 | 32kB | 1 | 50 | 96 | 64 | 64 | 32 | 16 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#24** | 16kB | 2 | 4kB | 4 | 256 | 128 | 64 | 16 | 8 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#25** | 32kB | 4 | 4kB | 1 | 256 | 96 | 64 | 32 | 16 | 64 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#26** | 32kB | 1 | 32kB | 2 | 96 | 96 | 64 | 16 | 64 | 40 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#27** | 32kB | 1 | 32kB | 2 | 64 | 128 | 16 | 8 | 64 | 128 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#28** | 64kB | 1 | 4kB | 4 | 50 | 96 | 16 | 64 | 8 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#29** | 64kB | 2 | 16kB | 4 | 128 | 96 | 64 | 8 | 32 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#30** | 16kB | 2 | 32kB | 4 | 128 | 96 | 64 | 64 | 8 | 16 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#31** | 32kB | 4 | 16kB | 4 | 96 | 96 | 32 | 16 | 16 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#32** | 16kB | 1 | 8kB | 1 | 256 | 256 | 64 | 32 | 8 | 64 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#33** | 64kB | 1 | 16kB | 2 | 256 | 96 | 16 | 8 | 16 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#34** | 16kB | 4 | 16kB | 1 | 64 | 256 | 64 | 8 | 16 | 32 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#35** | 16kB | 2 | 32kB | 1 | 256 | 256 | 32 | 32 | 8 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#36** | 64kB | 4 | 8kB | 2 | 256 | 96 | 64 | 32 | 64 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#37** | 16kB | 1 | 8kB | 2 | 256 | 128 | 32 | 16 | 32 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#38** | 16kB | 2 | 8kB | 1 | 256 | 128 | 16 | 16 | 8 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#39** | 64kB | 2 | 16kB | 2 | 64 | 128 | 32 | 32 | 16 | 128 | 2 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#40** | 16kB | 1 | 8kB | 1 | 256 | 128 | 16 | 64 | 32 | 128 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#41** | 32kB | 1 | 32kB | 4 | 96 | 96 | 64 | 32 | 32 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#42** | 64kB | 2 | 8kB | 2 | 96 | 128 | 64 | 64 | 8 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#43** | 64kB | 4 | 64kB | 2 | 64 | 256 | 32 | 64 | 32 | 16 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#44** | 16kB | 4 | 8kB | 2 | 50 | 128 | 32 | 64 | 16 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#45** | 16kB | 1 | 64kB | 1 | 50 | 96 | 32 | 16 | 64 | 40 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#46** | 16kB | 1 | 8kB | 2 | 50 | 256 | 32 | 16 | 16 | 128 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#47** | 64kB | 1 | 4kB | 1 | 64 | 256 | 16 | 32 | 32 | 16 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#48** | 16kB | 1 | 16kB | 1 | 256 | 96 | 32 | 32 | 8 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#49** | 32kB | 1 | 32kB | 4 | 64 | 128 | 16 | 64 | 64 | 128 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#50** | 64kB | 1 | 64kB | 2 | 64 | 96 | 32 | 16 | 64 | 64 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#51** | 64kB | 2 | 64kB | 1 | 64 | 256 | 32 | 32 | 32 | 16 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#52** | 32kB | 2 | 64kB | 4 | 128 | 96 | 16 | 16 | 8 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#53** | 32kB | 4 | 16kB | 4 | 50 | 128 | 64 | 8 | 64 | 128 | 2 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#54** | 32kB | 1 | 8kB | 1 | 64 | 128 | 16 | 8 | 8 | 128 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#55** | 16kB | 2 | 16kB | 2 | 64 | 128 | 16 | 64 | 8 | 64 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#56** | 16kB | 2 | 16kB | 1 | 256 | 96 | 64 | 8 | 16 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#57** | 16kB | 1 | 16kB | 4 | 96 | 128 | 32 | 32 | 16 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#58** | 32kB | 1 | 8kB | 2 | 50 | 256 | 32 | 16 | 8 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#59** | 64kB | 1 | 4kB | 4 | 128 | 96 | 64 | 16 | 16 | 16 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#60** | 16kB | 2 | 4kB | 1 | 50 | 128 | 16 | 32 | 16 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | p. 79 |
| **#61** | 64kB | 1 | 64kB | 1 | 128 | 256 | 64 | 16 | 32 | 16 | 2 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#62** | 64kB | 4 | 16kB | 1 | 128 | 256 | 64 | 16 | 64 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#63** | 32kB | 1 | 8kB | 1 | 64 | 256 | 32 | 16 | 64 | 40 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#64** | 32kB | 1 | 32kB | 1 | 128 | 128 | 16 | 32 | 8 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#65** | 16kB | 2 | 4kB | 4 | 64 | 128 | 16 | 32 | 64 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#66** | 32kB | 4 | 16kB | 4 | 64 | 96 | 16 | 64 | 32 | 16 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#67** | 64kB | 1 | 32kB | 4 | 128 | 256 | 32 | 64 | 16 | 64 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#68** | 32kB | 4 | 8kB | 4 | 96 | 256 | 32 | 16 | 16 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#69** | 64kB | 2 | 32kB | 4 | 50 | 96 | 16 | 32 | 8 | 64 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#70** | 32kB | 2 | 16kB | 1 | 128 | 128 | 16 | 8 | 16 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#71** | 64kB | 2 | 8kB | 4 | 50 | 96 | 64 | 8 | 8 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#72** | 64kB | 2 | 8kB | 4 | 64 | 256 | 64 | 16 | 16 | 16 | 3 | $2^{13}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#73** | 16kB | 2 | 4kB | 2 | 50 | 128 | 32 | 16 | 64 | 32 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#74** | 32kB | 4 | 16kB | 4 | 256 | 256 | 64 | 8 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#75** | 64kB | 2 | 64kB | 1 | 96 | 256 | 16 | 64 | 64 | 32 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#76** | 32kB | 4 | 4kB | 1 | 64 | 96 | 16 | 64 | 32 | 64 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#77** | 16kB | 2 | 16kB | 4 | 128 | 96 | 16 | 64 | 32 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#78** | 64kB | 1 | 4kB | 4 | 96 | 96 | 32 | 16 | 64 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#79** | 32kB | 2 | 32kB | 4 | 128 | 128 | 32 | 8 | 16 | 64 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#80** | 64kB | 4 | 4kB | 2 | 256 | 256 | 64 | 32 | 16 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#81** | 64kB | 4 | 4kB | 4 | 50 | 128 | 32 | 32 | 8 | 64 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#82** | 32kB | 1 | 8kB | 1 | 64 | 256 | 16 | 32 | 16 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#83** | 64kB | 2 | 8kB | 1 | 128 | 256 | 32 | 64 | 32 | 16 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#84** | 16kB | 1 | 4kB | 1 | 64 | 256 | 16 | 32 | 8 | 128 | 2 | $2^{11}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#85** | 32kB | 2 | 16kB | 2 | 128 | 128 | 16 | 8 | 8 | 128 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#86** | 64kB | 4 | 16kB | 1 | 128 | 96 | 32 | 8 | 32 | 16 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#87** | 32kB | 4 | 32kB | 1 | 96 | 128 | 32 | 64 | 32 | 128 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#88** | 32kB | 4 | 4kB | 2 | 256 | 128 | 64 | 64 | 32 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#89** | 16kB | 4 | 8kB | 1 | 96 | 256 | 16 | 16 | 16 | 16 | 3 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#90** | 16kB | 1 | 16kB | 1 | 64 | 96 | 64 | 32 | 32 | 16 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#91** | 64kB | 4 | 64kB | 4 | 96 | 256 | 64 | 16 | 8 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#92** | 64kB | 2 | 64kB | 4 | 128 | 128 | 16 | 32 | 8 | 64 | 2 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#93** | 16kB | 2 | 8kB | 2 | 128 | 256 | 16 | 16 | 16 | 16 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#94** | 32kB | 4 | 4kB | 1 | 64 | 96 | 16 | 32 | 16 | 32 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#95** | 32kB | 2 | 32kB | 2 | 256 | 128 | 64 | 32 | 32 | 128 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | p. 143 |
| **#96** | 64kB | 4 | 64kB | 4 | 50 | 256 | 16 | 16 | 8 | 40 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#97** | 16kB | 2 | 16kB | 4 | 128 | 96 | 64 | 16 | 32 | 64 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#98** | 64kB | 2 | 4kB | 4 | 256 | 128 | 32 | 64 | 8 | 40 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#99** | 16kB | 2 | 32kB | 2 | 64 | 128 | 16 | 32 | 64 | 32 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#100** | 16kB | 4 | 64kB | 2 | 96 | 128 | 16 | 16 | 32 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#101** | 16kB | 4 | 64kB | 2 | 96 | 256 | 16 | 64 | 8 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#102** | 16kB | 1 | 64kB | 2 | 64 | 256 | 16 | 16 | 64 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#103** | 32kB | 2 | 32kB | 2 | 256 | 96 | 16 | 8 | 64 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#104** | 32kB | 4 | 64kB | 1 | 128 | 96 | 16 | 64 | 32 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#105** | 16kB | 2 | 16kB | 4 | 96 | 96 | 16 | 8 | 32 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#106** | 64kB | 2 | 64kB | 2 | 64 | 128 | 32 | 8 | 16 | 64 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#107** | 64kB | 2 | 64kB | 2 | 256 | 96 | 64 | 64 | 16 | 32 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#108** | 16kB | 4 | 16kB | 1 | 128 | 96 | 64 | 32 | 32 | 128 | 3 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#109** | 64kB | 1 | 32kB | 4 | 96 | 128 | 16 | 16 | 64 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#110** | 64kB | 2 | 8kB | 1 | 96 | 96 | 64 | 8 | 16 | 40 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#111** | 32kB | 4 | 64kB | 2 | 256 | 128 | 16 | 8 | 64 | 64 | 2 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#112** | 16kB | 1 | 64kB | 2 | 64 | 256 | 32 | 64 | 16 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#113** | 32kB | 4 | 16kB | 2 | 256 | 96 | 16 | 64 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#114** | 64kB | 1 | 16kB | 2 | 128 | 96 | 16 | 8 | 8 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#115** | 32kB | 4 | 32kB | 4 | 256 | 96 | 16 | 8 | 64 | 64 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#116** | 16kB | 1 | 64kB | 4 | 50 | 96 | 16 | 32 | 64 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#117** | 32kB | 1 | 4kB | 2 | 128 | 256 | 64 | 64 | 8 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#118** | 64kB | 1 | 16kB | 1 | 128 | 128 | 16 | 32 | 64 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#119** | 32kB | 1 | 64kB | 4 | 128 | 256 | 32 | 8 | 64 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#120** | 32kB | 2 | 64kB | 4 | 64 | 256 | 16 | 8 | 64 | 128 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#121** | 16kB | 4 | 8kB | 2 | 64 | 96 | 32 | 8 | 64 | 32 | 2 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#122** | 32kB | 1 | 4kB | 1 | 64 | 128 | 64 | 32 | 64 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#123** | 32kB | 2 | 4kB | 4 | 50 | 96 | 16 | 16 | 16 | 16 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#124** | 64kB | 2 | 4kB | 4 | 50 | 128 | 32 | 32 | 8 | 32 | 1 | $2^{12}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#125** | 64kB | 1 | 4kB | 2 | 64 | 256 | 32 | 32 | 8 | 40 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#126** | 64kB | 4 | 8kB | 1 | 128 | 96 | 32 | 8 | 64 | 64 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#127** | 32kB | 2 | 16kB | 4 | 50 | 96 | 64 | 64 | 8 | 128 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#128** | 64kB | 1 | 8kB | 2 | 256 | 96 | 32 | 32 | 8 | 32 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#129** | 64kB | 2 | 64kB | 1 | 64 | 128 | 16 | 32 | 8 | 40 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#130** | 16kB | 4 | 32kB | 4 | 50 | 128 | 64 | 16 | 8 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#131** | 64kB | 1 | 4kB | 1 | 96 | 96 | 32 | 8 | 64 | 16 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#132** | 16kB | 4 | 8kB | 1 | 256 | 96 | 32 | 16 | 32 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#133** | 64kB | 4 | 64kB | 1 | 50 | 256 | 64 | 64 | 8 | 32 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#134** | 64kB | 4 | 64kB | 4 | 256 | 256 | 64 | 16 | 16 | 16 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#135** | 16kB | 2 | 8kB | 4 | 128 | 128 | 16 | 8 | 16 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#136** | 32kB | 1 | 64kB | 2 | 128 | 96 | 16 | 64 | 32 | 64 | 1 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#137** | 16kB | 2 | 16kB | 4 | 64 | 96 | 16 | 64 | 64 | 32 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#138** | 64kB | 2 | 16kB | 1 | 50 | 256 | 64 | 8 | 32 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#139** | 64kB | 4 | 16kB | 4 | 50 | 256 | 64 | 16 | 32 | 64 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#140** | 32kB | 2 | 8kB | 1 | 128 | 256 | 16 | 16 | 32 | 64 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#141** | 16kB | 2 | 64kB | 4 | 50 | 256 | 64 | 8 | 32 | 40 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#142** | 16kB | 1 | 4kB | 1 | 50 | 96 | 16 | 32 | 8 | 64 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | p. 142 |
| **#143** | 64kB | 1 | 32kB | 1 | 256 | 128 | 64 | 32 | 8 | 64 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#144** | 16kB | 4 | 64kB | 2 | 64 | 128 | 16 | 32 | 16 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#145** | 32kB | 1 | 4kB | 1 | 256 | 128 | 32 | 32 | 64 | 32 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#146** | 64kB | 2 | 4kB | 4 | 50 | 256 | 16 | 16 | 16 | 128 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#147** | 16kB | 1 | 16kB | 4 | 50 | 256 | 16 | 8 | 64 | 128 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#148** | 16kB | 1 | 64kB | 2 | 64 | 128 | 16 | 32 | 8 | 16 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#149** | 64kB | 4 | 8kB | 4 | 64 | 256 | 16 | 32 | 64 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#150** | 32kB | 1 | 4kB | 1 | 50 | 96 | 64 | 64 | 64 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#151** | 16kB | 2 | 8kB | 4 | 96 | 128 | 32 | 8 | 32 | 32 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#152** | 64kB | 4 | 64kB | 2 | 50 | 256 | 16 | 8 | 32 | 32 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#153** | 32kB | 4 | 64kB | 2 | 50 | 96 | 64 | 32 | 32 | 32 | 2 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#154** | 16kB | 4 | 64kB | 1 | 128 | 96 | 64 | 32 | 32 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#155** | 16kB | 2 | 16kB | 1 | 50 | 256 | 32 | 32 | 8 | 128 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#156** | 16kB | 1 | 8kB | 2 | 64 | 256 | 32 | 8 | 8 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | p. 143 |
| **#157** | 64kB | 2 | 8kB | 2 | 50 | 256 | 32 | 32 | 16 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#158** | 16kB | 2 | 8kB | 2 | 256 | 96 | 64 | 64 | 64 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#159** | 32kB | 2 | 64kB | 4 | 64 | 128 | 16 | 16 | 64 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#160** | 32kB | 4 | 16kB | 4 | 128 | 256 | 32 | 16 | 8 | 64 | 2 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#161** | 32kB | 4 | 64kB | 2 | 128 | 96 | 32 | 64 | 32 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | p. 143 |
| **#162** | 64kB | 2 | 32kB | 1 | 128 | 96 | 32 | 64 | 8 | 16 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#163** | 16kB | 2 | 8kB | 2 | 128 | 128 | 16 | 32 | 8 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#164** | 32kB | 2 | 32kB | 4 | 96 | 96 | 16 | 8 | 64 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#165** | 64kB | 4 | 16kB | 1 | 64 | 128 | 64 | 64 | 64 | 16 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#166** | 64kB | 2 | 32kB | 1 | 128 | 256 | 16 | 32 | 16 | 16 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#167** | 16kB | 2 | 32kB | 4 | 50 | 128 | 16 | 8 | 32 | 64 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#168** | 64kB | 2 | 16kB | 1 | 64 | 256 | 32 | 16 | 8 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#169** | 64kB | 2 | 16kB | 4 | 256 | 96 | 64 | 64 | 16 | 64 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#170** | 64kB | 1 | 16kB | 1 | 128 | 128 | 64 | 8 | 32 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#171** | 64kB | 1 | 8kB | 4 | 96 | 96 | 32 | 64 | 16 | 64 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#172** | 32kB | 1 | 8kB | 2 | 64 | 128 | 32 | 8 | 8 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#173** | 32kB | 4 | 16kB | 4 | 64 | 256 | 32 | 64 | 8 | 40 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#174** | 32kB | 2 | 32kB | 2 | 50 | 96 | 32 | 64 | 16 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#175** | 32kB | 4 | 4kB | 2 | 256 | 128 | 64 | 32 | 8 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#176** | 16kB | 1 | 64kB | 4 | 256 | 128 | 64 | 32 | 8 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#177** | 16kB | 1 | 16kB | 1 | 128 | 128 | 64 | 8 | 64 | 16 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#178** | 64kB | 4 | 8kB | 4 | 128 | 96 | 16 | 8 | 16 | 128 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#179** | 32kB | 1 | 4kB | 1 | 50 | 96 | 32 | 32 | 32 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#180** | 64kB | 4 | 4kB | 1 | 96 | 96 | 32 | 32 | 32 | 32 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#181** | 64kB | 2 | 32kB | 2 | 96 | 128 | 64 | 32 | 64 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#182** | 32kB | 2 | 32kB | 2 | 128 | 96 | 16 | 64 | 32 | 40 | 1 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#183** | 32kB | 2 | 16kB | 4 | 256 | 96 | 32 | 32 | 16 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#184** | 16kB | 1 | 4kB | 1 | 256 | 96 | 64 | 64 | 64 | 64 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#185** | 32kB | 1 | 4kB | 2 | 96 | 96 | 16 | 32 | 32 | 40 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#186** | 16kB | 4 | 4kB | 1 | 64 | 128 | 32 | 64 | 16 | 32 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#187** | 64kB | 1 | 16kB | 1 | 64 | 128 | 32 | 32 | 8 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#188** | 16kB | 4 | 32kB | 4 | 256 | 128 | 64 | 32 | 32 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#189** | 64kB | 2 | 8kB | 2 | 128 | 96 | 64 | 64 | 64 | 32 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#190** | 64kB | 1 | 16kB | 4 | 64 | 256 | 64 | 64 | 8 | 128 | 1 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#191** | 16kB | 4 | 8kB | 4 | 64 | 96 | 32 | 8 | 64 | 64 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#192** | 32kB | 4 | 16kB | 2 | 50 | 96 | 64 | 64 | 32 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#193** | 32kB | 1 | 4kB | 2 | 256 | 128 | 16 | 16 | 64 | 32 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#194** | 16kB | 1 | 8kB | 1 | 64 | 96 | 16 | 64 | 32 | 16 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#195** | 64kB | 1 | 16kB | 2 | 256 | 128 | 32 | 16 | 16 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#196** | 16kB | 4 | 64kB | 1 | 96 | 128 | 64 | 64 | 8 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#197** | 16kB | 2 | 64kB | 4 | 50 | 256 | 64 | 64 | 64 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#198** | 16kB | 2 | 4kB | 1 | 256 | 256 | 16 | 16 | 8 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#199** | 16kB | 4 | 32kB | 2 | 64 | 96 | 64 | 32 | 8 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#200** | 32kB | 1 | 4kB | 1 | 64 | 96 | 16 | 32 | 64 | 40 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#201** | 32kB | 1 | 32kB | 1 | 128 | 256 | 64 | 32 | 8 | 32 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#202** | 16kB | 2 | 8kB | 2 | 64 | 128 | 16 | 16 | 64 | 64 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#203** | 64kB | 1 | 64kB | 2 | 50 | 256 | 64 | 8 | 16 | 16 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#204** | 16kB | 4 | 8kB | 2 | 64 | 128 | 64 | 64 | 16 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#205** | 16kB | 2 | 4kB | 1 | 64 | 256 | 64 | 16 | 8 | 40 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#206** | 32kB | 4 | 16kB | 4 | 256 | 96 | 16 | 32 | 8 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#207** | 32kB | 1 | 64kB | 4 | 64 | 128 | 64 | 64 | 8 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#208** | 32kB | 4 | 16kB | 2 | 128 | 96 | 32 | 16 | 32 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#209** | 16kB | 2 | 8kB | 1 | 64 | 128 | 64 | 64 | 64 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#210** | 16kB | 4 | 32kB | 2 | 128 | 96 | 16 | 8 | 64 | 32 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#211** | 32kB | 2 | 8kB | 4 | 96 | 128 | 32 | 64 | 64 | 40 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#212** | 32kB | 4 | 16kB | 4 | 50 | 96 | 16 | 8 | 32 | 32 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#213** | 32kB | 1 | 64kB | 2 | 128 | 96 | 64 | 16 | 32 | 40 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#214** | 32kB | 2 | 64kB | 1 | 50 | 256 | 64 | 32 | 64 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#215** | 16kB | 1 | 32kB | 2 | 256 | 128 | 64 | 32 | 16 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#216** | 32kB | 2 | 64kB | 1 | 256 | 256 | 32 | 8 | 32 | 40 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#217** | 64kB | 2 | 64kB | 4 | 96 | 128 | 16 | 16 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#218** | 64kB | 2 | 16kB | 1 | 64 | 96 | 16 | 8 | 8 | 40 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#219** | 32kB | 1 | 8kB | 2 | 128 | 256 | 32 | 8 | 8 | 40 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#220** | 32kB | 2 | 4kB | 4 | 96 | 128 | 16 | 16 | 32 | 40 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#221** | 32kB | 4 | 64kB | 4 | 128 | 128 | 64 | 8 | 32 | 32 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#222** | 64kB | 1 | 16kB | 2 | 96 | 96 | 32 | 64 | 16 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#223** | 16kB | 4 | 32kB | 1 | 256 | 96 | 16 | 32 | 64 | 40 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#224** | 32kB | 2 | 4kB | 2 | 64 | 256 | 32 | 32 | 8 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#225** | 16kB | 4 | 16kB | 4 | 256 | 96 | 32 | 16 | 16 | 40 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#226** | 64kB | 4 | 16kB | 2 | 256 | 128 | 32 | 16 | 64 | 64 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#227** | 16kB | 2 | 4kB | 1 | 64 | 96 | 16 | 8 | 16 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#228** | 64kB | 4 | 64kB | 4 | 128 | 256 | 64 | 16 | 32 | 32 | 1 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#229** | 64kB | 2 | 32kB | 1 | 128 | 128 | 16 | 8 | 16 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#230** | 16kB | 2 | 4kB | 4 | 64 | 128 | 16 | 8 | 64 | 128 | 2 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#231** | 16kB | 4 | 64kB | 2 | 50 | 96 | 32 | 32 | 8 | 128 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#232** | 16kB | 2 | 16kB | 4 | 128 | 96 | 64 | 8 | 32 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#233** | 64kB | 2 | 4kB | 4 | 128 | 128 | 16 | 16 | 16 | 40 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#234** | 64kB | 2 | 64kB | 2 | 50 | 128 | 32 | 32 | 16 | 64 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#235** | 64kB | 2 | 8kB | 4 | 64 | 128 | 16 | 64 | 8 | 128 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#236** | 64kB | 2 | 4kB | 4 | 50 | 96 | 16 | 8 | 32 | 64 | 2 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#237** | 16kB | 1 | 8kB | 4 | 96 | 96 | 16 | 64 | 8 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#238** | 64kB | 2 | 8kB | 1 | 256 | 128 | 64 | 64 | 16 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#239** | 32kB | 2 | 4kB | 4 | 50 | 128 | 16 | 32 | 64 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#240** | 32kB | 2 | 16kB | 4 | 256 | 256 | 64 | 32 | 16 | 40 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#241** | 32kB | 4 | 16kB | 2 | 50 | 96 | 16 | 32 | 32 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#242** | 32kB | 4 | 8kB | 1 | 64 | 256 | 32 | 8 | 16 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#243** | 64kB | 4 | 8kB | 1 | 128 | 128 | 64 | 64 | 64 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#244** | 32kB | 4 | 32kB | 4 | 96 | 128 | 16 | 64 | 32 | 40 | 1 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#245** | 16kB | 1 | 16kB | 2 | 256 | 256 | 16 | 16 | 16 | 128 | 3 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#246** | 32kB | 4 | 8kB | 1 | 256 | 256 | 16 | 64 | 32 | 64 | 2 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#247** | 32kB | 2 | 16kB | 2 | 128 | 96 | 32 | 16 | 16 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#248** | 64kB | 4 | 64kB | 4 | 96 | 128 | 32 | 64 | 8 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#249** | 16kB | 1 | 8kB | 4 | 256 | 256 | 32 | 64 | 8 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#250** | 16kB | 4 | 4kB | 1 | 256 | 128 | 32 | 16 | 16 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#251** | 64kB | 2 | 64kB | 2 | 96 | 96 | 64 | 32 | 64 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#252** | 32kB | 2 | 16kB | 2 | 50 | 256 | 32 | 16 | 8 | 32 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#253** | 16kB | 4 | 16kB | 2 | 96 | 128 | 64 | 8 | 16 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#254** | 64kB | 1 | 8kB | 1 | 128 | 128 | 16 | 32 | 16 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#255** | 32kB | 4 | 4kB | 4 | 128 | 128 | 16 | 32 | 32 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#256** | 32kB | 1 | 4kB | 4 | 256 | 128 | 32 | 64 | 64 | 16 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#257** | 32kB | 1 | 4kB | 1 | 50 | 128 | 64 | 16 | 16 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#258** | 16kB | 2 | 8kB | 2 | 128 | 96 | 32 | 32 | 16 | 64 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#259** | 32kB | 4 | 64kB | 2 | 256 | 128 | 16 | 16 | 16 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#260** | 64kB | 1 | 16kB | 4 | 50 | 128 | 64 | 32 | 16 | 16 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#261** | 64kB | 2 | 32kB | 1 | 50 | 128 | 16 | 64 | 8 | 32 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#262** | 32kB | 1 | 64kB | 2 | 128 | 96 | 16 | 16 | 16 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#263** | 16kB | 4 | 64kB | 2 | 64 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#264** | 16kB | 2 | 16kB | 4 | 64 | 96 | 64 | 64 | 16 | 128 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#265** | 16kB | 2 | 32kB | 4 | 50 | 128 | 16 | 64 | 64 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#266** | 16kB | 4 | 16kB | 1 | 96 | 128 | 32 | 64 | 16 | 128 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#267** | 16kB | 1 | 8kB | 2 | 50 | 96 | 16 | 64 | 16 | 40 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#268** | 64kB | 1 | 4kB | 4 | 64 | 256 | 32 | 8 | 64 | 40 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#269** | 32kB | 4 | 64kB | 1 | 50 | 96 | 16 | 8 | 32 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#270** | 64kB | 2 | 16kB | 2 | 256 | 256 | 64 | 32 | 32 | 16 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#271** | 64kB | 4 | 4kB | 1 | 64 | 96 | 32 | 16 | 32 | 128 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#272** | 32kB | 4 | 4kB | 1 | 128 | 128 | 32 | 32 | 32 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#273** | 32kB | 4 | 64kB | 2 | 256 | 96 | 32 | 64 | 16 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#274** | 32kB | 4 | 4kB | 4 | 96 | 128 | 16 | 8 | 32 | 128 | 3 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#275** | 64kB | 2 | 8kB | 2 | 50 | 256 | 64 | 32 | 64 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#276** | 16kB | 1 | 64kB | 1 | 96 | 128 | 64 | 16 | 64 | 128 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#277** | 64kB | 1 | 32kB | 2 | 128 | 256 | 32 | 32 | 64 | 64 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#278** | 16kB | 2 | 8kB | 4 | 256 | 256 | 32 | 32 | 32 | 32 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#279** | 32kB | 1 | 64kB | 4 | 64 | 96 | 32 | 8 | 16 | 40 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#280** | 16kB | 2 | 32kB | 2 | 128 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#281** | 32kB | 1 | 16kB | 1 | 96 | 128 | 16 | 8 | 64 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#282** | 64kB | 1 | 32kB | 1 | 96 | 256 | 16 | 16 | 8 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#283** | 32kB | 4 | 8kB | 2 | 256 | 128 | 16 | 64 | 16 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#284** | 32kB | 1 | 32kB | 2 | 96 | 128 | 32 | 64 | 8 | 40 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#285** | 16kB | 1 | 16kB | 2 | 128 | 128 | 16 | 32 | 64 | 16 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#286** | 64kB | 1 | 16kB | 1 | 256 | 128 | 64 | 8 | 32 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#287** | 32kB | 4 | 64kB | 1 | 50 | 128 | 32 | 64 | 64 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#288** | 16kB | 2 | 16kB | 1 | 256 | 128 | 16 | 32 | 32 | 64 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#289** | 64kB | 2 | 32kB | 4 | 50 | 96 | 64 | 64 | 64 | 32 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#290** | 32kB | 4 | 16kB | 1 | 256 | 128 | 16 | 8 | 64 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#291** | 16kB | 1 | 8kB | 2 | 96 | 96 | 16 | 32 | 32 | 16 | 2 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#292** | 64kB | 1 | 64kB | 1 | 50 | 256 | 32 | 8 | 16 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#293** | 16kB | 1 | 8kB | 1 | 96 | 96 | 16 | 16 | 64 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#294** | 64kB | 2 | 64kB | 2 | 50 | 256 | 16 | 32 | 16 | 32 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#295** | 64kB | 1 | 4kB | 4 | 128 | 256 | 32 | 32 | 8 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#296** | 16kB | 1 | 64kB | 4 | 64 | 128 | 32 | 16 | 32 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#297** | 32kB | 2 | 64kB | 4 | 256 | 128 | 32 | 8 | 64 | 16 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#298** | 16kB | 4 | 64kB | 4 | 50 | 96 | 64 | 32 | 8 | 128 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#299** | 64kB | 4 | 4kB | 2 | 50 | 256 | 32 | 16 | 8 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#300** | 64kB | 1 | 32kB | 1 | 96 | 128 | 16 | 64 | 16 | 64 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#301** | 64kB | 1 | 4kB | 1 | 64 | 256 | 16 | 32 | 16 | 32 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#302** | 64kB | 4 | 64kB | 4 | 128 | 128 | 32 | 8 | 16 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#303** | 32kB | 2 | 64kB | 4 | 64 | 96 | 32 | 64 | 64 | 128 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#304** | 64kB | 4 | 32kB | 1 | 256 | 256 | 64 | 8 | 64 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#305** | 32kB | 4 | 32kB | 1 | 64 | 256 | 32 | 8 | 16 | 128 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#306** | 16kB | 4 | 16kB | 4 | 256 | 128 | 64 | 8 | 64 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#307** | 64kB | 2 | 64kB | 1 | 50 | 128 | 32 | 8 | 8 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#308** | 64kB | 1 | 16kB | 4 | 50 | 256 | 16 | 8 | 64 | 40 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#309** | 32kB | 4 | 64kB | 2 | 256 | 128 | 16 | 32 | 32 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#310** | 64kB | 2 | 8kB | 2 | 50 | 128 | 16 | 16 | 32 | 16 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#311** | 64kB | 4 | 8kB | 4 | 64 | 128 | 64 | 64 | 32 | 40 | 2 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#312** | 32kB | 4 | 32kB | 4 | 96 | 256 | 64 | 32 | 32 | 16 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#313** | 64kB | 4 | 64kB | 2 | 96 | 96 | 32 | 16 | 8 | 64 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#314** | 64kB | 4 | 32kB | 2 | 50 | 96 | 64 | 32 | 16 | 64 | 1 | $2^{13}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#315** | 32kB | 1 | 4kB | 2 | 96 | 128 | 32 | 16 | 32 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#316** | 32kB | 4 | 16kB | 4 | 50 | 256 | 64 | 16 | 8 | 32 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#317** | 64kB | 4 | 8kB | 1 | 256 | 96 | 32 | 32 | 32 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#318** | 32kB | 2 | 16kB | 2 | 128 | 128 | 32 | 32 | 32 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#319** | 64kB | 1 | 32kB | 4 | 256 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | p. 98 |
| **#320** | 16kB | 1 | 32kB | 2 | 128 | 128 | 64 | 32 | 64 | 64 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#321** | 32kB | 1 | 4kB | 4 | 50 | 128 | 16 | 64 | 16 | 40 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#322** | 16kB | 2 | 8kB | 2 | 128 | 128 | 16 | 8 | 64 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#323** | 16kB | 1 | 4kB | 1 | 128 | 256 | 16 | 32 | 64 | 32 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#324** | 16kB | 2 | 32kB | 4 | 50 | 96 | 64 | 8 | 32 | 128 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#325** | 64kB | 2 | 8kB | 1 | 96 | 96 | 32 | 32 | 16 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#326** | 32kB | 4 | 32kB | 4 | 128 | 96 | 64 | 64 | 16 | 64 | 2 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#327** | 64kB | 1 | 8kB | 1 | 64 | 96 | 64 | 64 | 16 | 40 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#328** | 64kB | 4 | 4kB | 2 | 50 | 128 | 64 | 8 | 64 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#329** | 32kB | 4 | 16kB | 4 | 64 | 128 | 16 | 8 | 8 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#330** | 64kB | 2 | 32kB | 4 | 256 | 128 | 32 | 16 | 8 | 32 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#331** | 16kB | 1 | 32kB | 4 | 50 | 96 | 64 | 32 | 16 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#332** | 32kB | 4 | 8kB | 4 | 64 | 128 | 64 | 8 | 64 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#333** | 64kB | 2 | 32kB | 2 | 256 | 96 | 16 | 64 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#334** | 64kB | 1 | 4kB | 1 | 128 | 96 | 16 | 32 | 16 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#335** | 32kB | 2 | 64kB | 1 | 128 | 256 | 32 | 64 | 64 | 32 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#336** | 16kB | 2 | 16kB | 4 | 64 | 128 | 16 | 8 | 16 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#337** | 32kB | 4 | 8kB | 1 | 256 | 96 | 16 | 64 | 16 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#338** | 16kB | 2 | 64kB | 2 | 50 | 128 | 32 | 8 | 32 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#339** | 64kB | 2 | 64kB | 2 | 96 | 128 | 64 | 32 | 64 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#340** | 16kB | 2 | 8kB | 4 | 256 | 96 | 64 | 16 | 16 | 32 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#341** | 16kB | 2 | 4kB | 1 | 256 | 96 | 16 | 16 | 8 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#342** | 32kB | 4 | 4kB | 2 | 256 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#343** | 64kB | 2 | 4kB | 1 | 96 | 96 | 16 | 32 | 8 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#344** | 32kB | 2 | 32kB | 1 | 256 | 256 | 32 | 8 | 8 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#345** | 64kB | 1 | 8kB | 4 | 128 | 128 | 64 | 64 | 32 | 16 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#346** | 32kB | 2 | 8kB | 2 | 64 | 96 | 16 | 32 | 16 | 32 | 3 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#347** | 32kB | 2 | 32kB | 4 | 96 | 128 | 32 | 64 | 64 | 128 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#348** | 64kB | 2 | 32kB | 1 | 128 | 128 | 64 | 32 | 64 | 16 | 3 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#349** | 32kB | 2 | 64kB | 2 | 50 | 256 | 32 | 16 | 8 | 128 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#350** | 64kB | 1 | 8kB | 1 | 128 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#351** | 16kB | 1 | 32kB | 4 | 50 | 128 | 64 | 16 | 16 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#352** | 64kB | 4 | 64kB | 2 | 128 | 128 | 16 | 64 | 32 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#353** | 64kB | 1 | 32kB | 2 | 128 | 256 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#354** | 32kB | 2 | 4kB | 2 | 96 | 128 | 64 | 32 | 64 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#355** | 64kB | 2 | 4kB | 4 | 128 | 256 | 16 | 32 | 8 | 64 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#356** | 64kB | 4 | 4kB | 2 | 64 | 256 | 32 | 16 | 16 | 40 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#357** | 16kB | 1 | 32kB | 4 | 64 | 96 | 16 | 64 | 16 | 32 | 2 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#358** | 32kB | 2 | 16kB | 2 | 50 | 96 | 16 | 32 | 16 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#359** | 16kB | 2 | 64kB | 2 | 128 | 128 | 32 | 16 | 16 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#360** | 64kB | 1 | 8kB | 2 | 256 | 256 | 32 | 8 | 16 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#361** | 64kB | 2 | 4kB | 2 | 50 | 96 | 16 | 64 | 8 | 32 | 1 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#362** | 32kB | 4 | 8kB | 4 | 256 | 128 | 32 | 16 | 8 | 128 | 2 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#363** | 32kB | 1 | 8kB | 2 | 256 | 128 | 64 | 8 | 8 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#364** | 32kB | 1 | 32kB | 4 | 256 | 128 | 64 | 16 | 64 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#365** | 16kB | 1 | 8kB | 1 | 50 | 96 | 64 | 32 | 32 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#366** | 32kB | 2 | 8kB | 1 | 50 | 256 | 16 | 64 | 16 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#367** | 32kB | 2 | 16kB | 2 | 128 | 256 | 16 | 32 | 64 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#368** | 32kB | 1 | 32kB | 2 | 50 | 256 | 16 | 64 | 8 | 40 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#369** | 16kB | 1 | 32kB | 4 | 64 | 256 | 32 | 8 | 8 | 16 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#370** | 64kB | 2 | 16kB | 4 | 64 | 96 | 32 | 8 | 8 | 64 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#371** | 64kB | 2 | 64kB | 1 | 96 | 96 | 32 | 16 | 8 | 32 | 3 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#372** | 64kB | 2 | 64kB | 2 | 50 | 96 | 32 | 8 | 8 | 40 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#373** | 16kB | 4 | 4kB | 1 | 256 | 96 | 32 | 32 | 64 | 128 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#374** | 32kB | 1 | 64kB | 4 | 96 | 256 | 32 | 64 | 8 | 32 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#375** | 32kB | 1 | 32kB | 2 | 128 | 96 | 64 | 16 | 32 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#376** | 64kB | 1 | 64kB | 4 | 128 | 256 | 32 | 16 | 8 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#377** | 16kB | 4 | 32kB | 2 | 64 | 128 | 64 | 16 | 8 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#378** | 32kB | 1 | 32kB | 2 | 256 | 128 | 64 | 16 | 16 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#379** | 64kB | 1 | 32kB | 1 | 50 | 256 | 32 | 32 | 32 | 32 | 3 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#380** | 32kB | 2 | 64kB | 4 | 64 | 256 | 64 | 64 | 64 | 16 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#381** | 64kB | 4 | 32kB | 2 | 96 | 96 | 16 | 64 | 32 | 64 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#382** | 64kB | 4 | 8kB | 4 | 96 | 128 | 32 | 64 | 8 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#383** | 32kB | 2 | 16kB | 2 | 128 | 256 | 32 | 8 | 32 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#384** | 16kB | 4 | 4kB | 4 | 50 | 96 | 16 | 64 | 64 | 128 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#385** | 32kB | 1 | 16kB | 1 | 64 | 256 | 32 | 16 | 16 | 128 | 3 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#386** | 32kB | 4 | 16kB | 2 | 256 | 256 | 64 | 32 | 64 | 40 | 1 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#387** | 32kB | 2 | 8kB | 4 | 128 | 256 | 64 | 64 | 8 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#388** | 64kB | 1 | 8kB | 4 | 96 | 128 | 32 | 64 | 64 | 32 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#389** | 64kB | 1 | 32kB | 2 | 256 | 128 | 64 | 16 | 64 | 16 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#390** | 64kB | 4 | 64kB | 2 | 128 | 128 | 64 | 16 | 16 | 40 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#391** | 64kB | 4 | 16kB | 2 | 256 | 256 | 16 | 32 | 32 | 16 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#392** | 16kB | 2 | 32kB | 2 | 50 | 96 | 32 | 16 | 8 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#393** | 64kB | 2 | 32kB | 2 | 64 | 256 | 16 | 8 | 32 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#394** | 32kB | 2 | 4kB | 4 | 256 | 256 | 32 | 8 | 64 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#395** | 64kB | 4 | 8kB | 4 | 128 | 96 | 32 | 32 | 8 | 16 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#396** | 64kB | 1 | 4kB | 1 | 64 | 256 | 16 | 64 | 32 | 40 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#397** | 32kB | 2 | 16kB | 1 | 64 | 128 | 32 | 32 | 16 | 40 | 1 | $2^{12}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#398** | 16kB | 4 | 8kB | 2 | 128 | 128 | 64 | 16 | 32 | 64 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#399** | 16kB | 1 | 8kB | 4 | 256 | 96 | 64 | 8 | 8 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#400** | 16kB | 1 | 64kB | 2 | 256 | 128 | 16 | 8 | 8 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#401** | 32kB | 4 | 32kB | 2 | 64 | 128 | 32 | 16 | 16 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#402** | 16kB | 4 | 4kB | 4 | 256 | 128 | 64 | 16 | 16 | 32 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#403** | 32kB | 1 | 4kB | 2 | 64 | 128 | 64 | 8 | 64 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#404** | 64kB | 4 | 16kB | 1 | 256 | 96 | 32 | 16 | 32 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#405** | 32kB | 1 | 16kB | 4 | 256 | 128 | 32 | 16 | 32 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#406** | 16kB | 1 | 4kB | 2 | 256 | 96 | 64 | 16 | 16 | 16 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#407** | 64kB | 4 | 64kB | 1 | 64 | 128 | 16 | 16 | 16 | 64 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#408** | 64kB | 1 | 16kB | 4 | 64 | 96 | 32 | 32 | 16 | 64 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#409** | 32kB | 2 | 16kB | 1 | 256 | 128 | 16 | 64 | 64 | 16 | 2 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#410** | 64kB | 4 | 8kB | 4 | 96 | 96 | 64 | 64 | 8 | 32 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#411** | 16kB | 2 | 8kB | 1 | 128 | 256 | 32 | 32 | 8 | 32 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#412** | 16kB | 1 | 4kB | 2 | 256 | 128 | 64 | 32 | 8 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#413** | 16kB | 2 | 64kB | 1 | 128 | 128 | 32 | 8 | 8 | 32 | 3 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#414** | 64kB | 2 | 4kB | 1 | 128 | 128 | 64 | 64 | 16 | 32 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#415** | 32kB | 4 | 8kB | 4 | 50 | 128 | 16 | 16 | 32 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#416** | 32kB | 4 | 32kB | 4 | 256 | 128 | 16 | 8 | 8 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#417** | 32kB | 2 | 4kB | 1 | 50 | 256 | 64 | 8 | 32 | 32 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | p. 79 |
| **#418** | 16kB | 1 | 16kB | 4 | 50 | 96 | 16 | 32 | 32 | 16 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#419** | 64kB | 4 | 4kB | 4 | 256 | 96 | 32 | 64 | 32 | 128 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#420** | 32kB | 4 | 64kB | 1 | 96 | 256 | 16 | 64 | 32 | 16 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#421** | 64kB | 2 | 64kB | 1 | 128 | 128 | 32 | 8 | 16 | 40 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#422** | 32kB | 4 | 16kB | 4 | 256 | 128 | 16 | 64 | 32 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#423** | 32kB | 2 | 8kB | 1 | 50 | 128 | 16 | 8 | 64 | 32 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#424** | 32kB | 2 | 16kB | 4 | 96 | 128 | 32 | 16 | 32 | 40 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | p. 166 |
| **#425** | 64kB | 1 | 16kB | 4 | 50 | 256 | 64 | 64 | 8 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#426** | 16kB | 2 | 4kB | 4 | 50 | 96 | 16 | 16 | 8 | 64 | 1 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#427** | 16kB | 4 | 4kB | 2 | 128 | 256 | 64 | 8 | 16 | 32 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#428** | 64kB | 4 | 8kB | 4 | 128 | 256 | 16 | 16 | 8 | 40 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#429** | 32kB | 4 | 8kB | 2 | 256 | 128 | 64 | 16 | 8 | 128 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#430** | 16kB | 4 | 8kB | 1 | 128 | 96 | 64 | 32 | 16 | 128 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#431** | 32kB | 2 | 64kB | 1 | 50 | 128 | 16 | 8 | 16 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#432** | 16kB | 4 | 64kB | 1 | 256 | 128 | 32 | 8 | 8 | 16 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#433** | 32kB | 2 | 4kB | 2 | 96 | 256 | 64 | 32 | 8 | 40 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#434** | 32kB | 4 | 32kB | 2 | 50 | 256 | 32 | 16 | 8 | 40 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#435** | 32kB | 2 | 64kB | 1 | 256 | 96 | 64 | 64 | 32 | 32 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#436** | 16kB | 1 | 16kB | 1 | 64 | 256 | 64 | 64 | 64 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#437** | 32kB | 4 | 32kB | 1 | 256 | 128 | 64 | 32 | 16 | 128 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#438** | 32kB | 2 | 8kB | 2 | 64 | 256 | 64 | 32 | 8 | 32 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#439** | 16kB | 1 | 8kB | 1 | 50 | 128 | 32 | 16 | 16 | 128 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#440** | 16kB | 2 | 64kB | 1 | 96 | 96 | 16 | 16 | 16 | 128 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#441** | 32kB | 2 | 8kB | 1 | 256 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#442** | 16kB | 2 | 8kB | 2 | 64 | 256 | 64 | 8 | 16 | 16 | 3 | $2^{11}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#443** | 16kB | 4 | 16kB | 2 | 128 | 256 | 32 | 32 | 8 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#444** | 64kB | 4 | 8kB | 2 | 50 | 256 | 32 | 64 | 64 | 128 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#445** | 16kB | 4 | 64kB | 1 | 256 | 128 | 16 | 8 | 16 | 128 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#446** | 64kB | 2 | 4kB | 4 | 96 | 256 | 64 | 16 | 64 | 16 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#447** | 32kB | 4 | 32kB | 1 | 96 | 128 | 16 | 16 | 16 | 64 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#448** | 16kB | 1 | 16kB | 1 | 64 | 256 | 64 | 64 | 8 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#449** | 64kB | 4 | 8kB | 1 | 64 | 256 | 32 | 32 | 32 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#450** | 32kB | 1 | 8kB | 1 | 256 | 128 | 64 | 8 | 64 | 40 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#451** | 16kB | 4 | 4kB | 1 | 96 | 96 | 16 | 64 | 16 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#452** | 32kB | 1 | 4kB | 1 | 256 | 96 | 64 | 64 | 32 | 40 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#453** | 32kB | 4 | 4kB | 2 | 50 | 256 | 64 | 16 | 8 | 40 | 2 | $2^{13}$ | 2 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#454** | 16kB | 2 | 16kB | 2 | 128 | 128 | 64 | 32 | 32 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#455** | 32kB | 2 | 64kB | 4 | 64 | 96 | 16 | 64 | 32 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#456** | 32kB | 2 | 16kB | 2 | 128 | 128 | 64 | 16 | 8 | 128 | 2 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#457** | 32kB | 1 | 8kB | 2 | 64 | 256 | 16 | 32 | 32 | 128 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#458** | 64kB | 2 | 8kB | 1 | 256 | 256 | 32 | 64 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#459** | 16kB | 1 | 4kB | 1 | 256 | 256 | 32 | 64 | 8 | 16 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#460** | 16kB | 2 | 8kB | 4 | 64 | 256 | 32 | 8 | 32 | 40 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#461** | 16kB | 4 | 8kB | 1 | 64 | 128 | 32 | 8 | 64 | 40 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#462** | 64kB | 2 | 16kB | 2 | 50 | 96 | 64 | 16 | 16 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#463** | 64kB | 2 | 32kB | 2 | 50 | 256 | 16 | 16 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#464** | 16kB | 2 | 64kB | 1 | 64 | 256 | 16 | 8 | 64 | 32 | 3 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#465** | 32kB | 2 | 8kB | 4 | 256 | 256 | 64 | 32 | 8 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#466** | 64kB | 4 | 64kB | 1 | 128 | 256 | 32 | 16 | 64 | 40 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#467** | 32kB | 2 | 4kB | 1 | 50 | 128 | 16 | 8 | 16 | 40 | 2 | $2^{11}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#468** | 64kB | 4 | 4kB | 1 | 96 | 96 | 64 | 16 | 64 | 40 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#469** | 16kB | 2 | 32kB | 1 | 50 | 128 | 32 | 64 | 32 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#470** | 64kB | 2 | 8kB | 2 | 256 | 128 | 64 | 8 | 8 | 64 | 3 | $2^{13}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#471** | 16kB | 1 | 4kB | 4 | 50 | 96 | 64 | 32 | 64 | 16 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#472** | 64kB | 1 | 32kB | 1 | 50 | 128 | 32 | 32 | 64 | 64 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#473** | 16kB | 1 | 4kB | 1 | 50 | 128 | 32 | 64 | 8 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#474** | 16kB | 4 | 8kB | 2 | 256 | 256 | 16 | 8 | 32 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#475** | 32kB | 4 | 64kB | 2 | 256 | 96 | 16 | 8 | 8 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#476** | 32kB | 2 | 4kB | 4 | 64 | 256 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#477** | 32kB | 1 | 16kB | 4 | 50 | 128 | 64 | 8 | 64 | 64 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#478** | 32kB | 4 | 4kB | 2 | 96 | 96 | 64 | 16 | 8 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#479** | 32kB | 1 | 4kB | 1 | 128 | 128 | 64 | 8 | 16 | 128 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#480** | 64kB | 2 | 8kB | 1 | 256 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#481** | 32kB | 2 | 32kB | 2 | 64 | 256 | 16 | 16 | 64 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#482** | 64kB | 2 | 64kB | 1 | 96 | 256 | 64 | 8 | 32 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#483** | 32kB | 4 | 8kB | 1 | 64 | 96 | 64 | 8 | 64 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#484** | 64kB | 1 | 32kB | 2 | 256 | 256 | 32 | 32 | 16 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#485** | 64kB | 4 | 32kB | 4 | 96 | 128 | 16 | 64 | 8 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#486** | 16kB | 2 | 16kB | 1 | 64 | 256 | 16 | 8 | 32 | 40 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#487** | 16kB | 2 | 8kB | 2 | 128 | 96 | 64 | 16 | 16 | 64 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#488** | 64kB | 1 | 16kB | 4 | 64 | 128 | 16 | 16 | 64 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#489** | 16kB | 4 | 16kB | 4 | 96 | 96 | 64 | 32 | 32 | 32 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#490** | 32kB | 1 | 4kB | 2 | 64 | 96 | 64 | 16 | 16 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#491** | 32kB | 4 | 8kB | 1 | 64 | 96 | 32 | 16 | 64 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#492** | 32kB | 1 | 4kB | 1 | 96 | 96 | 32 | 8 | 64 | 32 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#493** | 32kB | 4 | 4kB | 1 | 96 | 128 | 16 | 16 | 32 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#494** | 32kB | 4 | 32kB | 1 | 96 | 256 | 64 | 32 | 16 | 40 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#495** | 16kB | 2 | 32kB | 1 | 128 | 96 | 64 | 32 | 16 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#496** | 16kB | 1 | 32kB | 2 | 50 | 96 | 32 | 16 | 16 | 128 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#497** | 16kB | 1 | 32kB | 1 | 128 | 128 | 16 | 64 | 32 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#498** | 64kB | 4 | 32kB | 1 | 64 | 128 | 16 | 8 | 8 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#499** | 16kB | 4 | 64kB | 4 | 64 | 96 | 32 | 64 | 16 | 40 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#500** | 64kB | 4 | 8kB | 1 | 128 | 256 | 16 | 8 | 8 | 32 | 2 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#501** | 64kB | 1 | 8kB | 2 | 64 | 256 | 64 | 8 | 64 | 128 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#502** | 32kB | 2 | 16kB | 2 | 256 | 128 | 32 | 64 | 8 | 64 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#503** | 64kB | 1 | 4kB | 1 | 256 | 256 | 16 | 32 | 8 | 40 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#504** | 16kB | 1 | 64kB | 1 | 64 | 128 | 64 | 32 | 32 | 128 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#505** | 32kB | 2 | 64kB | 4 | 50 | 96 | 64 | 16 | 16 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#506** | 16kB | 2 | 4kB | 1 | 64 | 256 | 32 | 32 | 32 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#507** | 64kB | 4 | 16kB | 1 | 128 | 96 | 32 | 64 | 16 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#508** | 16kB | 1 | 32kB | 1 | 96 | 128 | 64 | 8 | 8 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#509** | 32kB | 1 | 4kB | 4 | 96 | 256 | 32 | 64 | 64 | 32 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#510** | 16kB | 2 | 64kB | 1 | 96 | 96 | 32 | 8 | 16 | 64 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#511** | 64kB | 2 | 64kB | 4 | 50 | 256 | 32 | 32 | 16 | 64 | 1 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#512** | 64kB | 4 | 4kB | 2 | 64 | 256 | 32 | 8 | 64 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#513** | 64kB | 2 | 4kB | 1 | 256 | 128 | 16 | 32 | 16 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#514** | 64kB | 1 | 32kB | 2 | 64 | 256 | 16 | 64 | 32 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#515** | 64kB | 2 | 4kB | 4 | 128 | 128 | 16 | 16 | 8 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#516** | 64kB | 1 | 64kB | 1 | 128 | 256 | 16 | 64 | 16 | 16 | 3 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#517** | 16kB | 4 | 64kB | 4 | 50 | 128 | 64 | 32 | 16 | 16 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#518** | 64kB | 2 | 64kB | 2 | 128 | 256 | 32 | 64 | 32 | 64 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | p. 132 |
| **#519** | 32kB | 2 | 4kB | 4 | 64 | 128 | 16 | 16 | 8 | 128 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#520** | 32kB | 2 | 16kB | 2 | 50 | 128 | 16 | 8 | 32 | 16 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#521** | 32kB | 4 | 8kB | 1 | 128 | 96 | 64 | 8 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#522** | 64kB | 4 | 4kB | 1 | 128 | 128 | 64 | 64 | 32 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#523** | 32kB | 1 | 16kB | 1 | 256 | 96 | 32 | 16 | 64 | 16 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#524** | 32kB | 2 | 8kB | 2 | 64 | 128 | 64 | 32 | 8 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#525** | 64kB | 2 | 16kB | 2 | 64 | 256 | 16 | 64 | 8 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#526** | 32kB | 1 | 16kB | 4 | 128 | 256 | 64 | 32 | 8 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#527** | 16kB | 2 | 32kB | 4 | 64 | 96 | 32 | 64 | 8 | 32 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#528** | 64kB | 4 | 8kB | 2 | 256 | 128 | 16 | 16 | 32 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#529** | 64kB | 1 | 8kB | 4 | 50 | 256 | 16 | 16 | 64 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#530** | 32kB | 1 | 4kB | 2 | 96 | 256 | 32 | 8 | 32 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#531** | 16kB | 4 | 8kB | 1 | 50 | 96 | 32 | 16 | 8 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#532** | 32kB | 2 | 32kB | 4 | 96 | 256 | 16 | 32 | 16 | 40 | 2 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#533** | 16kB | 2 | 32kB | 1 | 96 | 96 | 16 | 64 | 64 | 40 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#534** | 16kB | 4 | 16kB | 1 | 64 | 128 | 32 | 32 | 64 | 128 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#535** | 32kB | 1 | 8kB | 4 | 128 | 96 | 32 | 64 | 8 | 16 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#536** | 64kB | 4 | 8kB | 2 | 96 | 128 | 16 | 32 | 64 | 16 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#537** | 32kB | 1 | 4kB | 1 | 256 | 128 | 32 | 64 | 64 | 16 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#538** | 64kB | 1 | 32kB | 1 | 96 | 96 | 64 | 64 | 64 | 64 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#539** | 32kB | 4 | 8kB | 2 | 50 | 256 | 64 | 8 | 16 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#540** | 16kB | 2 | 16kB | 4 | 96 | 96 | 16 | 64 | 8 | 16 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#541** | 16kB | 4 | 4kB | 4 | 128 | 128 | 16 | 8 | 8 | 64 | 2 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#542** | 64kB | 2 | 16kB | 1 | 96 | 256 | 64 | 16 | 8 | 64 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#543** | 32kB | 4 | 64kB | 4 | 128 | 256 | 16 | 32 | 32 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#544** | 64kB | 4 | 16kB | 4 | 128 | 256 | 64 | 8 | 8 | 128 | 3 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#545** | 64kB | 2 | 8kB | 2 | 128 | 96 | 32 | 64 | 8 | 40 | 1 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#546** | 16kB | 4 | 64kB | 4 | 128 | 96 | 32 | 32 | 64 | 32 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#547** | 64kB | 2 | 4kB | 2 | 96 | 128 | 32 | 8 | 64 | 40 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#548** | 32kB | 2 | 8kB | 4 | 64 | 96 | 64 | 64 | 16 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#549** | 64kB | 1 | 64kB | 1 | 96 | 256 | 32 | 16 | 64 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#550** | 32kB | 1 | 8kB | 4 | 50 | 256 | 64 | 8 | 32 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#551** | 32kB | 4 | 4kB | 2 | 128 | 128 | 64 | 16 | 8 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#552** | 64kB | 1 | 32kB | 4 | 96 | 128 | 16 | 32 | 32 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#553** | 64kB | 1 | 64kB | 2 | 64 | 256 | 64 | 64 | 8 | 64 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#554** | 16kB | 4 | 64kB | 1 | 128 | 128 | 64 | 32 | 64 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#555** | 16kB | 4 | 16kB | 2 | 256 | 96 | 32 | 8 | 16 | 32 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#556** | 16kB | 1 | 64kB | 1 | 128 | 256 | 16 | 8 | 32 | 16 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#557** | 32kB | 2 | 8kB | 2 | 128 | 96 | 64 | 16 | 8 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#558** | 32kB | 2 | 16kB | 4 | 256 | 128 | 32 | 64 | 64 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#559** | 32kB | 4 | 32kB | 1 | 256 | 128 | 32 | 32 | 32 | 128 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#560** | 64kB | 2 | 8kB | 1 | 64 | 256 | 64 | 32 | 32 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#561** | 32kB | 2 | 4kB | 4 | 50 | 256 | 16 | 8 | 8 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#562** | 16kB | 1 | 32kB | 1 | 256 | 256 | 32 | 8 | 8 | 64 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#563** | 32kB | 2 | 32kB | 2 | 96 | 256 | 64 | 32 | 16 | 40 | 3 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#564** | 64kB | 4 | 64kB | 2 | 50 | 128 | 64 | 64 | 32 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#565** | 16kB | 4 | 32kB | 2 | 256 | 96 | 32 | 8 | 8 | 32 | 2 | $2^{10}$ | 1 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#566** | 32kB | 4 | 32kB | 1 | 50 | 96 | 32 | 16 | 16 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#567** | 64kB | 1 | 4kB | 1 | 128 | 96 | 64 | 64 | 64 | 16 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#568** | 64kB | 4 | 32kB | 2 | 64 | 128 | 32 | 64 | 8 | 128 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#569** | 32kB | 4 | 64kB | 2 | 256 | 96 | 16 | 16 | 16 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#570** | 64kB | 2 | 8kB | 2 | 50 | 128 | 64 | 16 | 8 | 16 | 2 | $2^{12}$ | 3 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#571** | 64kB | 2 | 4kB | 2 | 64 | 96 | 64 | 32 | 8 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#572** | 64kB | 4 | 16kB | 4 | 96 | 256 | 16 | 64 | 16 | 64 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#573** | 64kB | 2 | 8kB | 4 | 64 | 128 | 32 | 8 | 8 | 64 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#574** | 32kB | 2 | 32kB | 4 | 256 | 256 | 32 | 32 | 8 | 16 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#575** | 64kB | 1 | 16kB | 1 | 96 | 128 | 32 | 64 | 8 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#576** | 32kB | 1 | 64kB | 2 | 64 | 128 | 16 | 16 | 16 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#577** | 32kB | 4 | 8kB | 4 | 256 | 96 | 16 | 64 | 8 | 128 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#578** | 32kB | 4 | 4kB | 1 | 50 | 96 | 64 | 16 | 32 | 64 | 3 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#579** | 32kB | 4 | 32kB | 2 | 256 | 128 | 64 | 8 | 32 | 64 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#580** | 16kB | 2 | 32kB | 4 | 64 | 128 | 16 | 8 | 8 | 128 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#581** | 64kB | 1 | 16kB | 4 | 256 | 128 | 64 | 32 | 32 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#582** | 16kB | 2 | 4kB | 4 | 96 | 96 | 32 | 64 | 16 | 32 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#583** | 64kB | 2 | 8kB | 4 | 128 | 256 | 32 | 64 | 8 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#584** | 64kB | 2 | 64kB | 4 | 96 | 128 | 32 | 32 | 64 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#585** | 32kB | 4 | 8kB | 4 | 256 | 128 | 64 | 8 | 64 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#586** | 64kB | 4 | 32kB | 2 | 256 | 96 | 16 | 32 | 8 | 128 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#587** | 32kB | 2 | 32kB | 2 | 96 | 256 | 16 | 8 | 32 | 40 | 2 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#588** | 16kB | 1 | 4kB | 2 | 64 | 128 | 64 | 16 | 32 | 32 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#589** | 16kB | 4 | 32kB | 4 | 64 | 256 | 32 | 16 | 16 | 128 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#590** | 64kB | 2 | 8kB | 1 | 50 | 96 | 64 | 64 | 16 | 32 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#591** | 16kB | 4 | 4kB | 2 | 96 | 96 | 32 | 8 | 32 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#592** | 32kB | 4 | 4kB | 2 | 50 | 256 | 16 | 64 | 64 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#593** | 16kB | 2 | 4kB | 2 | 128 | 96 | 32 | 64 | 16 | 32 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#594** | 16kB | 1 | 8kB | 2 | 96 | 96 | 16 | 8 | 64 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#595** | 32kB | 4 | 16kB | 2 | 50 | 128 | 16 | 8 | 64 | 64 | 1 | $2^{11}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#596** | 32kB | 4 | 64kB | 1 | 64 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#597** | 16kB | 1 | 16kB | 4 | 96 | 128 | 16 | 32 | 16 | 64 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | p. 79 |
| **#598** | 64kB | 1 | 4kB | 4 | 128 | 128 | 16 | 16 | 64 | 40 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#599** | 16kB | 1 | 8kB | 4 | 64 | 128 | 16 | 8 | 16 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#600** | 16kB | 4 | 32kB | 4 | 50 | 128 | 16 | 16 | 32 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#601** | 32kB | 1 | 64kB | 4 | 256 | 128 | 32 | 8 | 32 | 32 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#602** | 64kB | 2 | 16kB | 2 | 96 | 96 | 16 | 16 | 32 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#603** | 32kB | 2 | 8kB | 2 | 256 | 128 | 16 | 8 | 8 | 16 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#604** | 16kB | 1 | 32kB | 1 | 96 | 128 | 16 | 32 | 16 | 16 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#605** | 32kB | 1 | 8kB | 2 | 256 | 128 | 16 | 32 | 16 | 16 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#606** | 32kB | 4 | 8kB | 2 | 64 | 96 | 16 | 8 | 16 | 16 | 1 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#607** | 64kB | 1 | 64kB | 4 | 50 | 128 | 32 | 32 | 32 | 40 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#608** | 64kB | 4 | 16kB | 1 | 128 | 128 | 32 | 8 | 64 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#609** | 16kB | 4 | 16kB | 2 | 256 | 128 | 64 | 32 | 64 | 16 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#610** | 64kB | 4 | 16kB | 1 | 256 | 96 | 16 | 8 | 64 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#611** | 64kB | 1 | 8kB | 1 | 50 | 96 | 32 | 32 | 64 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#612** | 32kB | 2 | 8kB | 2 | 256 | 96 | 16 | 64 | 32 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#613** | 64kB | 1 | 16kB | 1 | 96 | 256 | 16 | 16 | 64 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#614** | 64kB | 4 | 32kB | 4 | 256 | 256 | 64 | 32 | 32 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#615** | 16kB | 1 | 64kB | 1 | 96 | 256 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#616** | 64kB | 4 | 32kB | 1 | 96 | 96 | 64 | 32 | 8 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#617** | 64kB | 1 | 4kB | 1 | 64 | 96 | 32 | 64 | 32 | 40 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#618** | 64kB | 4 | 32kB | 1 | 64 | 128 | 32 | 32 | 64 | 16 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#619** | 32kB | 4 | 4kB | 4 | 256 | 256 | 16 | 32 | 16 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#620** | 16kB | 2 | 4kB | 1 | 128 | 96 | 64 | 8 | 16 | 64 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#621** | 64kB | 2 | 32kB | 1 | 64 | 128 | 32 | 64 | 8 | 32 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#622** | 64kB | 4 | 16kB | 4 | 96 | 96 | 16 | 8 | 16 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#623** | 16kB | 1 | 32kB | 2 | 96 | 128 | 64 | 64 | 8 | 40 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#624** | 32kB | 2 | 32kB | 2 | 50 | 256 | 32 | 8 | 8 | 64 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#625** | 16kB | 4 | 4kB | 2 | 256 | 128 | 16 | 64 | 64 | 40 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#626** | 16kB | 1 | 8kB | 1 | 128 | 96 | 32 | 64 | 64 | 32 | 2 | $2^{11}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#627** | 32kB | 2 | 64kB | 4 | 256 | 128 | 16 | 32 | 8 | 128 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#628** | 16kB | 1 | 8kB | 4 | 64 | 96 | 32 | 64 | 8 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#629** | 16kB | 1 | 4kB | 2 | 128 | 128 | 64 | 32 | 32 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#630** | 64kB | 2 | 32kB | 2 | 128 | 128 | 16 | 16 | 32 | 64 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#631** | 16kB | 4 | 8kB | 2 | 256 | 96 | 32 | 16 | 8 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#632** | 32kB | 2 | 8kB | 4 | 50 | 128 | 64 | 8 | 8 | 64 | 1 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#633** | 16kB | 2 | 4kB | 2 | 50 | 96 | 32 | 8 | 16 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#634** | 16kB | 2 | 64kB | 1 | 128 | 128 | 16 | 16 | 64 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#635** | 64kB | 2 | 32kB | 4 | 96 | 128 | 64 | 32 | 8 | 16 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#636** | 32kB | 2 | 8kB | 2 | 50 | 256 | 16 | 8 | 8 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#637** | 32kB | 1 | 32kB | 2 | 128 | 96 | 64 | 64 | 32 | 16 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#638** | 16kB | 2 | 4kB | 1 | 50 | 256 | 32 | 32 | 64 | 40 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#639** | 64kB | 4 | 4kB | 1 | 96 | 96 | 32 | 64 | 8 | 128 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#640** | 16kB | 2 | 4kB | 4 | 50 | 128 | 16 | 64 | 8 | 64 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#641** | 16kB | 4 | 32kB | 1 | 96 | 128 | 64 | 16 | 8 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#642** | 16kB | 1 | 16kB | 2 | 50 | 128 | 16 | 16 | 64 | 40 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#643** | 64kB | 4 | 8kB | 2 | 256 | 256 | 64 | 8 | 16 | 32 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#644** | 32kB | 2 | 64kB | 1 | 256 | 256 | 32 | 32 | 16 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#645** | 16kB | 1 | 16kB | 1 | 128 | 256 | 16 | 32 | 16 | 128 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#646** | 64kB | 1 | 32kB | 1 | 128 | 96 | 32 | 32 | 32 | 128 | 2 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#647** | 64kB | 1 | 32kB | 4 | 50 | 96 | 32 | 32 | 8 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#648** | 32kB | 2 | 4kB | 1 | 128 | 96 | 16 | 16 | 64 | 64 | 2 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#649** | 16kB | 4 | 16kB | 2 | 50 | 96 | 64 | 8 | 64 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#650** | 32kB | 4 | 32kB | 1 | 64 | 96 | 16 | 64 | 16 | 16 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#651** | 32kB | 4 | 32kB | 4 | 128 | 128 | 32 | 16 | 8 | 16 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#652** | 32kB | 2 | 4kB | 2 | 96 | 128 | 32 | 8 | 32 | 64 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#653** | 16kB | 4 | 8kB | 2 | 96 | 256 | 32 | 16 | 64 | 64 | 2 | $2^{14}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#654** | 64kB | 1 | 16kB | 2 | 128 | 256 | 32 | 16 | 32 | 16 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#655** | 64kB | 2 | 16kB | 2 | 256 | 256 | 64 | 32 | 16 | 40 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#656** | 64kB | 2 | 16kB | 1 | 128 | 256 | 64 | 32 | 32 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#657** | 64kB | 1 | 32kB | 1 | 64 | 128 | 64 | 8 | 8 | 64 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#658** | 64kB | 2 | 32kB | 2 | 256 | 128 | 64 | 64 | 16 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#659** | 16kB | 2 | 64kB | 4 | 256 | 128 | 16 | 16 | 64 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#660** | 32kB | 1 | 4kB | 4 | 50 | 128 | 16 | 64 | 16 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#661** | 16kB | 4 | 32kB | 4 | 256 | 256 | 64 | 64 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#662** | 64kB | 1 | 8kB | 4 | 64 | 96 | 32 | 64 | 8 | 32 | 1 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#663** | 32kB | 1 | 16kB | 2 | 128 | 128 | 16 | 64 | 16 | 32 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#664** | 64kB | 4 | 32kB | 1 | 256 | 128 | 64 | 64 | 32 | 40 | 1 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#665** | 16kB | 1 | 32kB | 4 | 96 | 96 | 16 | 16 | 32 | 128 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#666** | 16kB | 1 | 8kB | 2 | 50 | 128 | 32 | 32 | 8 | 32 | 3 | $2^{12}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#667** | 32kB | 2 | 8kB | 2 | 128 | 256 | 32 | 32 | 16 | 128 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#668** | 16kB | 4 | 8kB | 2 | 64 | 128 | 32 | 64 | 16 | 16 | 2 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#669** | 32kB | 1 | 16kB | 1 | 50 | 128 | 64 | 64 | 8 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#670** | 32kB | 1 | 16kB | 2 | 128 | 128 | 64 | 64 | 8 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#671** | 32kB | 2 | 32kB | 4 | 128 | 96 | 64 | 32 | 32 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | p. 79 |
| **#672** | 16kB | 1 | 64kB | 1 | 64 | 256 | 32 | 32 | 32 | 16 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#673** | 16kB | 2 | 8kB | 4 | 256 | 128 | 16 | 64 | 8 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#674** | 32kB | 1 | 64kB | 2 | 64 | 96 | 32 | 64 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#675** | 64kB | 1 | 32kB | 4 | 96 | 128 | 32 | 8 | 8 | 16 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#676** | 32kB | 2 | 8kB | 4 | 256 | 128 | 16 | 32 | 64 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#677** | 32kB | 4 | 64kB | 2 | 128 | 96 | 64 | 8 | 64 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#678** | 16kB | 1 | 8kB | 1 | 128 | 96 | 16 | 8 | 16 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#679** | 64kB | 2 | 4kB | 2 | 50 | 256 | 32 | 16 | 16 | 128 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#680** | 32kB | 1 | 16kB | 1 | 128 | 128 | 32 | 64 | 8 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#681** | 32kB | 1 | 4kB | 2 | 50 | 128 | 32 | 16 | 32 | 32 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#682** | 16kB | 1 | 8kB | 2 | 256 | 128 | 64 | 8 | 64 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#683** | 64kB | 1 | 64kB | 4 | 256 | 256 | 32 | 32 | 8 | 32 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#684** | 16kB | 1 | 8kB | 1 | 128 | 96 | 32 | 8 | 16 | 64 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#685** | 64kB | 4 | 16kB | 4 | 128 | 256 | 32 | 32 | 16 | 40 | 1 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#686** | 64kB | 1 | 16kB | 4 | 96 | 256 | 16 | 16 | 64 | 128 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#687** | 16kB | 2 | 64kB | 4 | 96 | 128 | 64 | 32 | 8 | 128 | 1 | $2^{11}$ | 3 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#688** | 32kB | 1 | 64kB | 2 | 50 | 128 | 16 | 16 | 32 | 64 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#689** | 32kB | 1 | 32kB | 4 | 96 | 128 | 64 | 16 | 8 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#690** | 32kB | 1 | 64kB | 4 | 64 | 96 | 32 | 64 | 16 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#691** | 16kB | 4 | 32kB | 1 | 256 | 96 | 64 | 32 | 8 | 16 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#692** | 64kB | 4 | 64kB | 4 | 96 | 96 | 32 | 8 | 32 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#693** | 16kB | 4 | 64kB | 1 | 256 | 96 | 16 | 64 | 64 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#694** | 64kB | 2 | 32kB | 2 | 256 | 96 | 64 | 64 | 8 | 32 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#695** | 16kB | 1 | 8kB | 2 | 256 | 256 | 32 | 64 | 32 | 16 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#696** | 64kB | 4 | 64kB | 2 | 256 | 128 | 64 | 64 | 16 | 40 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#697** | 64kB | 4 | 64kB | 1 | 96 | 128 | 16 | 32 | 64 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#698** | 16kB | 1 | 64kB | 1 | 64 | 256 | 16 | 8 | 16 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#699** | 32kB | 2 | 32kB | 1 | 128 | 128 | 64 | 16 | 16 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#700** | 64kB | 4 | 16kB | 1 | 50 | 96 | 32 | 64 | 16 | 32 | 2 | $2^{14}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#701** | 64kB | 1 | 8kB | 2 | 64 | 128 | 32 | 64 | 16 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#702** | 16kB | 4 | 64kB | 2 | 64 | 96 | 32 | 8 | 32 | 32 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#703** | 16kB | 2 | 32kB | 1 | 256 | 256 | 64 | 64 | 32 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#704** | 16kB | 1 | 64kB | 2 | 50 | 96 | 16 | 8 | 64 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#705** | 32kB | 1 | 16kB | 2 | 50 | 96 | 64 | 16 | 64 | 32 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#706** | 32kB | 4 | 64kB | 1 | 256 | 256 | 32 | 16 | 32 | 32 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#707** | 32kB | 2 | 16kB | 4 | 256 | 96 | 32 | 64 | 8 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#708** | 64kB | 4 | 32kB | 1 | 64 | 128 | 64 | 8 | 8 | 64 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#709** | 16kB | 2 | 64kB | 1 | 64 | 128 | 16 | 16 | 16 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#710** | 16kB | 4 | 8kB | 4 | 96 | 96 | 32 | 16 | 64 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#711** | 64kB | 2 | 16kB | 4 | 50 | 256 | 64 | 32 | 8 | 64 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#712** | 16kB | 2 | 16kB | 4 | 64 | 128 | 32 | 64 | 32 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#713** | 32kB | 2 | 16kB | 2 | 128 | 256 | 32 | 64 | 64 | 128 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | p. 79 |
| **#714** | 32kB | 1 | 8kB | 1 | 128 | 256 | 32 | 16 | 32 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#715** | 64kB | 2 | 8kB | 4 | 96 | 128 | 32 | 32 | 8 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#716** | 32kB | 1 | 4kB | 1 | 50 | 128 | 32 | 8 | 64 | 128 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#717** | 16kB | 4 | 8kB | 4 | 96 | 256 | 16 | 64 | 64 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#718** | 64kB | 4 | 64kB | 1 | 128 | 128 | 64 | 32 | 32 | 32 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#719** | 32kB | 1 | 64kB | 2 | 96 | 128 | 16 | 32 | 8 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#720** | 32kB | 1 | 8kB | 4 | 128 | 256 | 16 | 8 | 64 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#721** | 16kB | 1 | 8kB | 4 | 256 | 128 | 64 | 8 | 32 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#722** | 64kB | 4 | 16kB | 4 | 128 | 96 | 32 | 16 | 8 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#723** | 64kB | 4 | 8kB | 4 | 128 | 256 | 32 | 8 | 16 | 40 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#724** | 32kB | 1 | 32kB | 4 | 50 | 128 | 32 | 64 | 16 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#725** | 64kB | 2 | 64kB | 4 | 256 | 96 | 32 | 16 | 32 | 40 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#726** | 16kB | 1 | 16kB | 4 | 64 | 128 | 16 | 32 | 64 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#727** | 64kB | 1 | 64kB | 1 | 64 | 256 | 16 | 8 | 32 | 128 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#728** | 64kB | 1 | 64kB | 1 | 64 | 128 | 64 | 32 | 16 | 40 | 2 | $2^{13}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#729** | 32kB | 4 | 8kB | 4 | 128 | 96 | 16 | 64 | 8 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#730** | 16kB | 2 | 32kB | 4 | 64 | 96 | 16 | 32 | 8 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#731** | 16kB | 4 | 64kB | 2 | 64 | 128 | 32 | 16 | 16 | 128 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#732** | 64kB | 4 | 16kB | 4 | 128 | 128 | 64 | 32 | 64 | 64 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | p. 143 |
| **#733** | 16kB | 4 | 32kB | 2 | 128 | 256 | 32 | 8 | 8 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#734** | 16kB | 4 | 16kB | 2 | 96 | 256 | 32 | 16 | 8 | 128 | 1 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#735** | 32kB | 1 | 4kB | 2 | 128 | 96 | 16 | 16 | 32 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#736** | 32kB | 4 | 32kB | 1 | 128 | 96 | 32 | 16 | 32 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#737** | 16kB | 2 | 8kB | 1 | 50 | 96 | 32 | 16 | 16 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#738** | 32kB | 4 | 16kB | 2 | 96 | 128 | 32 | 64 | 64 | 16 | 2 | $2^{11}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#739** | 16kB | 2 | 16kB | 1 | 96 | 128 | 32 | 8 | 16 | 32 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#740** | 64kB | 4 | 8kB | 1 | 64 | 256 | 16 | 8 | 32 | 128 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#741** | 64kB | 4 | 64kB | 4 | 64 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#742** | 16kB | 2 | 32kB | 4 | 256 | 128 | 64 | 64 | 32 | 40 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#743** | 32kB | 4 | 32kB | 1 | 50 | 256 | 16 | 32 | 16 | 64 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#744** | 32kB | 2 | 16kB | 2 | 96 | 128 | 32 | 16 | 64 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#745** | 16kB | 1 | 64kB | 2 | 96 | 128 | 16 | 32 | 16 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#746** | 64kB | 2 | 4kB | 4 | 96 | 128 | 32 | 32 | 16 | 64 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#747** | 32kB | 2 | 4kB | 4 | 64 | 96 | 32 | 32 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#748** | 16kB | 2 | 32kB | 1 | 50 | 128 | 64 | 16 | 16 | 128 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#749** | 64kB | 4 | 32kB | 2 | 64 | 96 | 16 | 16 | 32 | 16 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#750** | 32kB | 1 | 4kB | 1 | 50 | 128 | 32 | 32 | 64 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#751** | 64kB | 2 | 16kB | 2 | 256 | 256 | 16 | 64 | 16 | 16 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#752** | 16kB | 4 | 32kB | 1 | 50 | 128 | 32 | 64 | 32 | 40 | 1 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#753** | 64kB | 1 | 32kB | 1 | 96 | 128 | 16 | 32 | 8 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#754** | 32kB | 1 | 8kB | 2 | 128 | 256 | 16 | 8 | 16 | 64 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#755** | 16kB | 2 | 8kB | 4 | 96 | 256 | 16 | 8 | 16 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#756** | 64kB | 2 | 16kB | 2 | 128 | 128 | 16 | 32 | 8 | 16 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#757** | 64kB | 4 | 16kB | 1 | 256 | 128 | 64 | 32 | 8 | 32 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#758** | 32kB | 2 | 32kB | 4 | 96 | 128 | 64 | 16 | 16 | 64 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#759** | 16kB | 2 | 4kB | 1 | 64 | 256 | 16 | 64 | 16 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#760** | 64kB | 4 | 16kB | 2 | 256 | 256 | 32 | 8 | 16 | 40 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#761** | 64kB | 1 | 16kB | 4 | 128 | 128 | 64 | 32 | 64 | 64 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#762** | 32kB | 2 | 32kB | 2 | 96 | 256 | 16 | 8 | 64 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#763** | 16kB | 1 | 32kB | 4 | 64 | 128 | 16 | 32 | 8 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#764** | 16kB | 1 | 4kB | 2 | 96 | 96 | 64 | 64 | 64 | 64 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#765** | 64kB | 4 | 8kB | 4 | 64 | 256 | 16 | 8 | 16 | 16 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#766** | 16kB | 2 | 16kB | 4 | 256 | 256 | 16 | 8 | 8 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#767** | 16kB | 4 | 16kB | 2 | 64 | 256 | 16 | 32 | 16 | 16 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#768** | 32kB | 1 | 64kB | 4 | 96 | 96 | 64 | 32 | 32 | 16 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#769** | 32kB | 2 | 4kB | 4 | 128 | 96 | 32 | 16 | 64 | 40 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#770** | 16kB | 1 | 4kB | 4 | 256 | 128 | 32 | 8 | 8 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#771** | 16kB | 4 | 32kB | 2 | 64 | 96 | 32 | 64 | 32 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#772** | 64kB | 1 | 4kB | 2 | 256 | 256 | 64 | 32 | 32 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#773** | 32kB | 1 | 4kB | 1 | 256 | 128 | 16 | 32 | 8 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#774** | 64kB | 2 | 8kB | 2 | 50 | 128 | 64 | 8 | 64 | 40 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#775** | 32kB | 1 | 32kB | 2 | 64 | 96 | 64 | 16 | 16 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#776** | 64kB | 4 | 64kB | 2 | 128 | 96 | 32 | 16 | 8 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#777** | 32kB | 2 | 64kB | 2 | 96 | 96 | 16 | 16 | 8 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | p. 110 |
| **#778** | 64kB | 4 | 4kB | 4 | 256 | 256 | 64 | 32 | 8 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#779** | 16kB | 4 | 4kB | 2 | 50 | 256 | 32 | 64 | 64 | 64 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#780** | 32kB | 4 | 64kB | 4 | 96 | 256 | 16 | 8 | 16 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#781** | 64kB | 1 | 16kB | 2 | 128 | 96 | 32 | 32 | 8 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#782** | 16kB | 2 | 64kB | 1 | 64 | 256 | 64 | 64 | 32 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#783** | 64kB | 4 | 4kB | 4 | 256 | 256 | 32 | 64 | 64 | 16 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#784** | 16kB | 4 | 64kB | 4 | 128 | 256 | 16 | 8 | 8 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#785** | 16kB | 4 | 32kB | 4 | 64 | 256 | 32 | 64 | 32 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#786** | 32kB | 4 | 8kB | 4 | 96 | 96 | 64 | 32 | 64 | 128 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#787** | 16kB | 1 | 8kB | 4 | 256 | 96 | 32 | 8 | 64 | 32 | 1 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#788** | 16kB | 4 | 4kB | 2 | 256 | 128 | 32 | 16 | 8 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#789** | 64kB | 2 | 8kB | 4 | 64 | 96 | 64 | 8 | 8 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#790** | 16kB | 1 | 4kB | 4 | 96 | 96 | 32 | 32 | 16 | 64 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#791** | 16kB | 1 | 4kB | 1 | 128 | 256 | 16 | 16 | 32 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#792** | 64kB | 4 | 8kB | 4 | 256 | 128 | 32 | 32 | 32 | 64 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#793** | 64kB | 2 | 4kB | 1 | 50 | 96 | 64 | 8 | 16 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#794** | 32kB | 2 | 16kB | 2 | 64 | 128 | 64 | 8 | 8 | 32 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#795** | 64kB | 4 | 16kB | 1 | 256 | 256 | 64 | 16 | 64 | 40 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#796** | 32kB | 1 | 32kB | 1 | 50 | 128 | 64 | 16 | 8 | 16 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#797** | 64kB | 4 | 32kB | 4 | 96 | 96 | 16 | 32 | 8 | 128 | 3 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#798** | 32kB | 4 | 4kB | 4 | 50 | 96 | 16 | 32 | 32 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#799** | 64kB | 2 | 8kB | 4 | 64 | 128 | 16 | 32 | 32 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#800** | 64kB | 4 | 8kB | 4 | 256 | 96 | 64 | 16 | 16 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#801** | 32kB | 2 | 4kB | 2 | 256 | 128 | 16 | 64 | 8 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#802** | 32kB | 1 | 16kB | 1 | 50 | 128 | 16 | 32 | 64 | 32 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#803** | 64kB | 2 | 16kB | 1 | 64 | 128 | 64 | 64 | 32 | 16 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#804** | 64kB | 4 | 64kB | 1 | 96 | 96 | 32 | 32 | 8 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#805** | 64kB | 1 | 16kB | 2 | 96 | 128 | 32 | 16 | 32 | 64 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#806** | 32kB | 1 | 32kB | 2 | 50 | 128 | 32 | 32 | 16 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#807** | 32kB | 4 | 4kB | 4 | 64 | 256 | 64 | 8 | 8 | 128 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#808** | 64kB | 2 | 64kB | 1 | 96 | 128 | 64 | 64 | 16 | 32 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#809** | 32kB | 1 | 32kB | 4 | 64 | 256 | 16 | 16 | 8 | 64 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#810** | 32kB | 1 | 32kB | 1 | 64 | 128 | 32 | 8 | 64 | 32 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#811** | 64kB | 1 | 16kB | 2 | 64 | 96 | 32 | 8 | 64 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#812** | 64kB | 2 | 32kB | 2 | 50 | 128 | 32 | 32 | 16 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#813** | 32kB | 4 | 64kB | 2 | 256 | 96 | 16 | 32 | 64 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#814** | 32kB | 2 | 32kB | 2 | 96 | 128 | 64 | 64 | 32 | 16 | 3 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#815** | 64kB | 2 | 16kB | 2 | 128 | 128 | 32 | 16 | 8 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#816** | 32kB | 2 | 8kB | 1 | 50 | 96 | 64 | 8 | 16 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#817** | 16kB | 1 | 32kB | 2 | 64 | 128 | 64 | 64 | 8 | 16 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#818** | 32kB | 4 | 64kB | 2 | 256 | 256 | 32 | 16 | 64 | 40 | 1 | $2^{13}$ | 3 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#819** | 32kB | 2 | 8kB | 2 | 128 | 128 | 64 | 32 | 8 | 128 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#820** | 32kB | 1 | 4kB | 4 | 50 | 256 | 64 | 8 | 32 | 64 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#821** | 32kB | 2 | 32kB | 1 | 64 | 128 | 32 | 32 | 32 | 40 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | p. 143 |
| **#822** | 32kB | 2 | 4kB | 4 | 64 | 256 | 64 | 8 | 64 | 128 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#823** | 16kB | 2 | 4kB | 1 | 64 | 128 | 32 | 8 | 64 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#824** | 32kB | 1 | 64kB | 4 | 128 | 128 | 64 | 16 | 32 | 16 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#825** | 16kB | 2 | 64kB | 4 | 96 | 96 | 64 | 64 | 16 | 32 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#826** | 64kB | 1 | 64kB | 4 | 128 | 96 | 32 | 8 | 16 | 128 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#827** | 32kB | 1 | 64kB | 2 | 50 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#828** | 32kB | 4 | 32kB | 1 | 64 | 128 | 16 | 32 | 16 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#829** | 32kB | 4 | 64kB | 4 | 64 | 96 | 16 | 16 | 16 | 16 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#830** | 64kB | 1 | 64kB | 2 | 50 | 128 | 32 | 32 | 16 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#831** | 32kB | 1 | 16kB | 1 | 256 | 96 | 32 | 16 | 64 | 40 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#832** | 64kB | 4 | 4kB | 2 | 50 | 128 | 16 | 64 | 64 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#833** | 16kB | 4 | 64kB | 4 | 96 | 96 | 64 | 8 | 32 | 40 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#834** | 64kB | 4 | 64kB | 4 | 64 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#835** | 64kB | 1 | 32kB | 4 | 50 | 256 | 32 | 32 | 32 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#836** | 32kB | 1 | 4kB | 2 | 96 | 128 | 64 | 8 | 16 | 64 | 1 | $2^{12}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#837** | 64kB | 1 | 8kB | 4 | 64 | 96 | 32 | 64 | 8 | 64 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#838** | 16kB | 2 | 64kB | 2 | 256 | 128 | 64 | 64 | 32 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#839** | 32kB | 2 | 4kB | 2 | 64 | 128 | 16 | 16 | 64 | 64 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#840** | 64kB | 2 | 32kB | 4 | 128 | 128 | 64 | 64 | 32 | 128 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#841** | 64kB | 2 | 16kB | 1 | 128 | 128 | 64 | 16 | 64 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#842** | 64kB | 1 | 32kB | 2 | 50 | 256 | 16 | 8 | 64 | 64 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#843** | 16kB | 1 | 32kB | 4 | 256 | 256 | 16 | 64 | 16 | 32 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#844** | 64kB | 4 | 8kB | 2 | 96 | 256 | 32 | 32 | 32 | 32 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#845** | 64kB | 4 | 32kB | 4 | 96 | 256 | 32 | 8 | 64 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#846** | 16kB | 4 | 16kB | 2 | 50 | 96 | 32 | 8 | 16 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#847** | 16kB | 2 | 32kB | 4 | 256 | 96 | 32 | 32 | 32 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#848** | 32kB | 4 | 8kB | 4 | 96 | 96 | 32 | 64 | 16 | 32 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#849** | 16kB | 4 | 8kB | 2 | 96 | 96 | 32 | 32 | 16 | 16 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#850** | 32kB | 4 | 32kB | 2 | 96 | 256 | 64 | 64 | 16 | 16 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#851** | 32kB | 2 | 32kB | 2 | 128 | 96 | 32 | 16 | 16 | 128 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#852** | 32kB | 4 | 8kB | 2 | 128 | 128 | 32 | 16 | 16 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#853** | 16kB | 2 | 8kB | 1 | 96 | 256 | 64 | 32 | 32 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#854** | 32kB | 4 | 64kB | 4 | 64 | 128 | 64 | 8 | 8 | 64 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#855** | 32kB | 1 | 8kB | 2 | 96 | 256 | 64 | 32 | 8 | 16 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#856** | 64kB | 4 | 16kB | 1 | 128 | 128 | 64 | 8 | 16 | 16 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#857** | 64kB | 2 | 64kB | 2 | 96 | 128 | 32 | 16 | 8 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#858** | 16kB | 4 | 64kB | 4 | 64 | 256 | 32 | 64 | 8 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#859** | 16kB | 4 | 4kB | 4 | 64 | 96 | 16 | 16 | 16 | 16 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#860** | 16kB | 4 | 64kB | 4 | 128 | 96 | 32 | 32 | 64 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#861** | 64kB | 1 | 16kB | 1 | 50 | 128 | 32 | 32 | 32 | 64 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#862** | 64kB | 2 | 16kB | 1 | 128 | 96 | 64 | 16 | 32 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#863** | 16kB | 2 | 16kB | 4 | 256 | 128 | 32 | 8 | 16 | 16 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#864** | 64kB | 4 | 16kB | 4 | 96 | 128 | 16 | 16 | 16 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#865** | 64kB | 1 | 4kB | 1 | 256 | 128 | 16 | 8 | 8 | 64 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#866** | 64kB | 4 | 16kB | 4 | 50 | 256 | 16 | 32 | 8 | 16 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#867** | 64kB | 2 | 4kB | 4 | 256 | 96 | 32 | 16 | 32 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#868** | 64kB | 4 | 64kB | 2 | 96 | 96 | 32 | 64 | 64 | 128 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#869** | 16kB | 2 | 64kB | 2 | 96 | 256 | 16 | 8 | 64 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#870** | 64kB | 2 | 64kB | 2 | 96 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#871** | 32kB | 4 | 8kB | 1 | 256 | 128 | 64 | 32 | 8 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#872** | 64kB | 1 | 16kB | 4 | 96 | 96 | 64 | 32 | 8 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#873** | 16kB | 4 | 16kB | 1 | 50 | 256 | 16 | 32 | 64 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#874** | 64kB | 1 | 32kB | 4 | 96 | 96 | 32 | 16 | 8 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#875** | 32kB | 4 | 64kB | 4 | 256 | 96 | 16 | 32 | 16 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#876** | 16kB | 2 | 64kB | 2 | 256 | 128 | 16 | 32 | 64 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#877** | 64kB | 2 | 8kB | 4 | 50 | 96 | 64 | 16 | 64 | 128 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#878** | 16kB | 1 | 16kB | 2 | 256 | 128 | 16 | 32 | 32 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#879** | 64kB | 2 | 16kB | 2 | 64 | 96 | 32 | 64 | 64 | 16 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#880** | 32kB | 2 | 16kB | 1 | 96 | 96 | 64 | 64 | 64 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#881** | 32kB | 1 | 32kB | 1 | 50 | 256 | 64 | 64 | 64 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#882** | 64kB | 4 | 4kB | 1 | 96 | 96 | 64 | 32 | 16 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#883** | 64kB | 1 | 8kB | 4 | 128 | 256 | 16 | 32 | 16 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#884** | 32kB | 1 | 4kB | 4 | 256 | 256 | 16 | 32 | 32 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#885** | 16kB | 2 | 64kB | 2 | 128 | 256 | 64 | 64 | 32 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#886** | 32kB | 1 | 4kB | 4 | 64 | 96 | 16 | 32 | 16 | 40 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#887** | 16kB | 1 | 64kB | 1 | 96 | 256 | 32 | 8 | 64 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#888** | 32kB | 4 | 4kB | 1 | 128 | 96 | 64 | 16 | 8 | 32 | 1 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#889** | 16kB | 1 | 4kB | 4 | 96 | 128 | 32 | 8 | 32 | 32 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#890** | 32kB | 2 | 8kB | 1 | 64 | 256 | 16 | 64 | 32 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#891** | 64kB | 2 | 32kB | 1 | 128 | 256 | 32 | 32 | 8 | 128 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#892** | 32kB | 2 | 4kB | 4 | 256 | 128 | 32 | 8 | 16 | 40 | 1 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#893** | 32kB | 4 | 32kB | 2 | 50 | 128 | 16 | 64 | 16 | 32 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#894** | 32kB | 4 | 4kB | 1 | 50 | 256 | 32 | 32 | 8 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#895** | 64kB | 1 | 4kB | 1 | 64 | 128 | 32 | 64 | 32 | 40 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#896** | 32kB | 2 | 32kB | 2 | 128 | 128 | 32 | 8 | 64 | 40 | 1 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#897** | 16kB | 1 | 64kB | 2 | 64 | 96 | 16 | 32 | 32 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#898** | 16kB | 2 | 4kB | 2 | 128 | 96 | 32 | 64 | 16 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#899** | 32kB | 4 | 64kB | 4 | 50 | 256 | 16 | 64 | 64 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#900** | 16kB | 1 | 4kB | 4 | 50 | 128 | 64 | 64 | 8 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#901** | 16kB | 1 | 64kB | 4 | 256 | 96 | 32 | 16 | 32 | 40 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#902** | 32kB | 1 | 8kB | 2 | 96 | 96 | 16 | 32 | 64 | 16 | 1 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#903** | 32kB | 2 | 4kB | 2 | 96 | 256 | 16 | 64 | 32 | 16 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#904** | 32kB | 2 | 32kB | 2 | 96 | 256 | 16 | 64 | 32 | 40 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#905** | 64kB | 4 | 16kB | 1 | 128 | 256 | 64 | 16 | 8 | 16 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#906** | 32kB | 1 | 64kB | 4 | 96 | 96 | 16 | 16 | 64 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#907** | 64kB | 1 | 8kB | 4 | 50 | 256 | 32 | 16 | 64 | 40 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#908** | 64kB | 4 | 16kB | 1 | 128 | 128 | 16 | 8 | 8 | 40 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#909** | 16kB | 4 | 64kB | 2 | 96 | 128 | 32 | 16 | 64 | 64 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#910** | 32kB | 1 | 8kB | 1 | 256 | 128 | 64 | 32 | 16 | 32 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#911** | 16kB | 1 | 32kB | 4 | 50 | 96 | 32 | 16 | 8 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#912** | 64kB | 2 | 16kB | 4 | 96 | 128 | 32 | 8 | 64 | 128 | 2 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#913** | 32kB | 2 | 8kB | 1 | 64 | 128 | 16 | 64 | 32 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#914** | 64kB | 1 | 16kB | 2 | 96 | 96 | 32 | 16 | 8 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#915** | 16kB | 4 | 32kB | 4 | 50 | 96 | 64 | 8 | 16 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#916** | 16kB | 4 | 64kB | 4 | 96 | 128 | 16 | 64 | 16 | 40 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#917** | 64kB | 1 | 4kB | 2 | 96 | 96 | 64 | 16 | 64 | 64 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#918** | 16kB | 2 | 16kB | 1 | 50 | 256 | 32 | 32 | 16 | 128 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#919** | 32kB | 2 | 8kB | 4 | 64 | 256 | 32 | 16 | 8 | 16 | 1 | $2^{11}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#920** | 16kB | 4 | 8kB | 1 | 256 | 128 | 16 | 16 | 16 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#921** | 16kB | 1 | 32kB | 2 | 50 | 128 | 16 | 8 | 64 | 64 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#922** | 64kB | 1 | 64kB | 2 | 96 | 128 | 64 | 64 | 64 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#923** | 16kB | 1 | 64kB | 4 | 96 | 256 | 32 | 8 | 8 | 128 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#924** | 64kB | 4 | 64kB | 4 | 50 | 96 | 64 | 64 | 8 | 32 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#925** | 32kB | 1 | 4kB | 4 | 50 | 96 | 16 | 16 | 16 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#926** | 16kB | 4 | 4kB | 4 | 64 | 96 | 64 | 16 | 32 | 128 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#927** | 16kB | 4 | 16kB | 1 | 50 | 256 | 32 | 16 | 8 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#928** | 16kB | 1 | 4kB | 1 | 256 | 256 | 64 | 8 | 8 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#929** | 16kB | 1 | 64kB | 1 | 50 | 96 | 64 | 16 | 64 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#930** | 16kB | 2 | 64kB | 2 | 128 | 128 | 64 | 16 | 16 | 128 | 2 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#931** | 64kB | 4 | 16kB | 1 | 64 | 128 | 32 | 8 | 32 | 32 | 2 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#932** | 16kB | 1 | 32kB | 2 | 256 | 256 | 64 | 8 | 8 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#933** | 16kB | 2 | 4kB | 2 | 96 | 96 | 16 | 32 | 8 | 32 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#934** | 64kB | 4 | 16kB | 4 | 256 | 96 | 32 | 64 | 64 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#935** | 32kB | 1 | 32kB | 1 | 50 | 96 | 32 | 32 | 32 | 40 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#936** | 64kB | 4 | 32kB | 4 | 50 | 256 | 32 | 16 | 32 | 32 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#937** | 32kB | 4 | 8kB | 1 | 50 | 256 | 64 | 16 | 16 | 40 | 2 | $2^{11}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#938** | 16kB | 2 | 32kB | 4 | 256 | 256 | 32 | 16 | 16 | 64 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#939** | 32kB | 4 | 8kB | 1 | 128 | 128 | 16 | 16 | 64 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#940** | 32kB | 4 | 32kB | 4 | 256 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#941** | 64kB | 1 | 8kB | 1 | 50 | 128 | 32 | 64 | 8 | 16 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#942** | 64kB | 1 | 64kB | 4 | 50 | 96 | 16 | 64 | 8 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#943** | 32kB | 2 | 4kB | 1 | 96 | 128 | 32 | 64 | 32 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#944** | 64kB | 1 | 16kB | 4 | 256 | 96 | 32 | 16 | 8 | 128 | 1 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#945** | 32kB | 1 | 32kB | 1 | 256 | 256 | 16 | 16 | 64 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#946** | 16kB | 2 | 4kB | 2 | 128 | 128 | 64 | 64 | 8 | 64 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#947** | 32kB | 4 | 32kB | 4 | 256 | 256 | 64 | 64 | 8 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#948** | 64kB | 4 | 32kB | 2 | 50 | 128 | 64 | 64 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#949** | 64kB | 1 | 8kB | 4 | 96 | 128 | 64 | 8 | 32 | 64 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#950** | 32kB | 2 | 16kB | 2 | 50 | 96 | 16 | 16 | 32 | 16 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#951** | 16kB | 1 | 8kB | 1 | 256 | 256 | 16 | 8 | 8 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#952** | 32kB | 4 | 64kB | 2 | 64 | 96 | 16 | 32 | 8 | 64 | 2 | $2^{14}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#953** | 16kB | 2 | 8kB | 4 | 50 | 128 | 64 | 16 | 16 | 16 | 2 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#954** | 32kB | 4 | 64kB | 2 | 128 | 96 | 64 | 8 | 32 | 32 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#955** | 32kB | 4 | 64kB | 1 | 64 | 96 | 16 | 32 | 64 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#956** | 16kB | 1 | 8kB | 4 | 96 | 256 | 32 | 64 | 32 | 16 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#957** | 16kB | 2 | 32kB | 2 | 256 | 96 | 16 | 8 | 64 | 32 | 1 | $2^{14}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#958** | 64kB | 4 | 16kB | 1 | 256 | 128 | 64 | 32 | 8 | 128 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#959** | 16kB | 2 | 16kB | 2 | 64 | 128 | 64 | 32 | 16 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#960** | 32kB | 2 | 8kB | 1 | 256 | 256 | 64 | 8 | 16 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#961** | 32kB | 4 | 16kB | 4 | 256 | 128 | 32 | 8 | 16 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#962** | 64kB | 4 | 4kB | 4 | 96 | 128 | 64 | 16 | 32 | 40 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#963** | 16kB | 2 | 16kB | 4 | 128 | 96 | 64 | 32 | 16 | 40 | 2 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | p. 98 |
| **#964** | 16kB | 1 | 32kB | 1 | 64 | 256 | 16 | 8 | 64 | 16 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#965** | 16kB | 2 | 16kB | 1 | 128 | 128 | 16 | 32 | 64 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#966** | 64kB | 2 | 8kB | 1 | 50 | 256 | 64 | 64 | 64 | 32 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#967** | 16kB | 2 | 32kB | 4 | 64 | 96 | 64 | 8 | 8 | 64 | 3 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#968** | 32kB | 4 | 32kB | 2 | 256 | 96 | 32 | 32 | 32 | 64 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#969** | 64kB | 4 | 32kB | 1 | 128 | 128 | 32 | 64 | 64 | 128 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | p. 142 |
| **#970** | 16kB | 1 | 8kB | 2 | 128 | 96 | 16 | 16 | 32 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#971** | 16kB | 2 | 8kB | 1 | 50 | 256 | 32 | 64 | 64 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#972** | 64kB | 4 | 32kB | 2 | 256 | 96 | 64 | 64 | 8 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#973** | 32kB | 1 | 8kB | 4 | 256 | 256 | 32 | 16 | 8 | 32 | 2 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#974** | 32kB | 1 | 32kB | 4 | 128 | 256 | 64 | 64 | 64 | 32 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#975** | 16kB | 2 | 16kB | 4 | 256 | 256 | 64 | 16 | 32 | 32 | 2 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#976** | 32kB | 4 | 32kB | 2 | 96 | 256 | 32 | 32 | 8 | 40 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#977** | 64kB | 4 | 32kB | 2 | 64 | 256 | 32 | 8 | 32 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#978** | 32kB | 1 | 8kB | 1 | 50 | 128 | 32 | 16 | 64 | 40 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#979** | 64kB | 2 | 4kB | 2 | 256 | 128 | 64 | 16 | 64 | 64 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#980** | 16kB | 2 | 4kB | 1 | 96 | 256 | 32 | 16 | 64 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#981** | 16kB | 2 | 32kB | 2 | 128 | 128 | 32 | 64 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#982** | 64kB | 1 | 8kB | 1 | 64 | 96 | 32 | 8 | 16 | 40 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#983** | 16kB | 4 | 8kB | 4 | 128 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#984** | 64kB | 2 | 32kB | 1 | 256 | 256 | 16 | 8 | 32 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#985** | 64kB | 1 | 64kB | 4 | 128 | 256 | 32 | 8 | 8 | 64 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#986** | 16kB | 1 | 8kB | 2 | 128 | 96 | 64 | 32 | 16 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#987** | 16kB | 4 | 4kB | 2 | 256 | 128 | 64 | 8 | 16 | 32 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#988** | 16kB | 2 | 4kB | 4 | 50 | 128 | 16 | 8 | 8 | 40 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | p. 161 |
| **#989** | 32kB | 4 | 32kB | 1 | 128 | 128 | 64 | 32 | 16 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#990** | 16kB | 1 | 16kB | 1 | 96 | 128 | 16 | 8 | 8 | 16 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#991** | 64kB | 4 | 4kB | 4 | 96 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#992** | 32kB | 4 | 64kB | 1 | 256 | 96 | 64 | 64 | 16 | 128 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#993** | 16kB | 1 | 64kB | 1 | 96 | 256 | 16 | 8 | 16 | 64 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#994** | 64kB | 2 | 64kB | 4 | 256 | 96 | 64 | 8 | 16 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#995** | 16kB | 4 | 4kB | 2 | 64 | 128 | 64 | 8 | 32 | 16 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#996** | 32kB | 2 | 16kB | 1 | 64 | 256 | 64 | 16 | 8 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#997** | 16kB | 2 | 16kB | 4 | 64 | 96 | 64 | 16 | 8 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#998** | 32kB | 2 | 4kB | 1 | 96 | 128 | 32 | 16 | 8 | 64 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#999** | 16kB | 2 | 32kB | 1 | 50 | 96 | 64 | 8 | 32 | 32 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1000** | 32kB | 1 | 8kB | 1 | 256 | 128 | 32 | 64 | 32 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1001** | 32kB | 1 | 32kB | 1 | 50 | 128 | 16 | 32 | 64 | 32 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1002** | 16kB | 1 | 16kB | 1 | 50 | 128 | 32 | 16 | 16 | 128 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1003** | 16kB | 1 | 32kB | 2 | 64 | 96 | 64 | 16 | 8 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1004** | 64kB | 2 | 8kB | 4 | 96 | 256 | 32 | 32 | 32 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1005** | 16kB | 2 | 64kB | 4 | 128 | 96 | 64 | 16 | 32 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1006** | 16kB | 2 | 16kB | 1 | 128 | 256 | 64 | 64 | 16 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1007** | 32kB | 4 | 32kB | 4 | 64 | 128 | 32 | 16 | 8 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1008** | 16kB | 1 | 4kB | 1 | 256 | 96 | 32 | 32 | 16 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1009** | 64kB | 2 | 8kB | 2 | 50 | 96 | 16 | 32 | 8 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1010** | 32kB | 2 | 16kB | 1 | 128 | 256 | 16 | 8 | 32 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1011** | 16kB | 4 | 16kB | 2 | 50 | 256 | 64 | 8 | 32 | 32 | 1 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1012** | 16kB | 1 | 16kB | 1 | 96 | 256 | 64 | 16 | 64 | 64 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1013** | 32kB | 2 | 8kB | 2 | 64 | 128 | 16 | 8 | 8 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1014** | 16kB | 2 | 64kB | 1 | 256 | 128 | 32 | 16 | 64 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1015** | 64kB | 2 | 32kB | 1 | 128 | 128 | 16 | 16 | 64 | 128 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1016** | 64kB | 2 | 64kB | 4 | 96 | 96 | 16 | 32 | 8 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1017** | 16kB | 2 | 16kB | 1 | 96 | 128 | 16 | 32 | 8 | 40 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1018** | 32kB | 4 | 16kB | 2 | 64 | 256 | 64 | 8 | 8 | 128 | 3 | $2^{11}$ | 2 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1019** | 32kB | 1 | 4kB | 4 | 128 | 256 | 32 | 64 | 64 | 64 | 1 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1020** | 16kB | 4 | 16kB | 4 | 50 | 256 | 16 | 16 | 32 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1021** | 16kB | 1 | 4kB | 2 | 128 | 128 | 32 | 32 | 8 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1022** | 32kB | 1 | 8kB | 1 | 64 | 128 | 16 | 64 | 64 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1023** | 64kB | 2 | 32kB | 4 | 64 | 128 | 32 | 64 | 8 | 64 | 1 | $2^{10}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#$2^{10}$** | 32kB | 1 | 32kB | 4 | 50 | 96 | 32 | 16 | 32 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1025** | 16kB | 4 | 32kB | 4 | 50 | 96 | 16 | 16 | 64 | 64 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1026** | 32kB | 2 | 32kB | 4 | 256 | 256 | 64 | 32 | 8 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1027** | 64kB | 4 | 32kB | 4 | 128 | 96 | 64 | 16 | 16 | 16 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1028** | 16kB | 4 | 16kB | 4 | 256 | 128 | 32 | 64 | 64 | 16 | 1 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1029** | 32kB | 4 | 4kB | 4 | 64 | 256 | 64 | 64 | 32 | 32 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1030** | 16kB | 1 | 8kB | 2 | 96 | 128 | 16 | 32 | 16 | 128 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | p. 110 |
| **#1031** | 32kB | 4 | 8kB | 1 | 256 | 128 | 32 | 16 | 32 | 128 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1032** | 16kB | 1 | 4kB | 1 | 64 | 256 | 32 | 8 | 32 | 128 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1033** | 64kB | 2 | 32kB | 4 | 96 | 128 | 64 | 32 | 8 | 64 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1034** | 16kB | 1 | 64kB | 1 | 50 | 96 | 64 | 32 | 32 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1035** | 64kB | 2 | 32kB | 4 | 256 | 256 | 16 | 16 | 8 | 64 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1036** | 32kB | 2 | 8kB | 4 | 256 | 256 | 64 | 64 | 64 | 128 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1037** | 64kB | 1 | 8kB | 2 | 96 | 256 | 32 | 16 | 8 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1038** | 16kB | 2 | 32kB | 2 | 256 | 256 | 16 | 64 | 16 | 40 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1039** | 32kB | 2 | 16kB | 1 | 64 | 128 | 64 | 16 | 16 | 32 | 1 | $2^{14}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1040** | 64kB | 1 | 64kB | 4 | 64 | 256 | 32 | 16 | 32 | 32 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1041** | 16kB | 4 | 16kB | 1 | 50 | 256 | 32 | 8 | 16 | 32 | 2 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1042** | 16kB | 2 | 8kB | 1 | 64 | 96 | 64 | 16 | 64 | 40 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1043** | 32kB | 1 | 4kB | 4 | 64 | 256 | 32 | 32 | 16 | 32 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1044** | 32kB | 1 | 8kB | 1 | 128 | 96 | 64 | 8 | 32 | 32 | 3 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1045** | 32kB | 4 | 4kB | 2 | 128 | 96 | 64 | 64 | 16 | 128 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1046** | 64kB | 2 | 32kB | 4 | 96 | 96 | 32 | 64 | 8 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1047** | 16kB | 4 | 64kB | 4 | 64 | 96 | 32 | 64 | 16 | 64 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1048** | 32kB | 1 | 64kB | 1 | 128 | 128 | 32 | 16 | 32 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1049** | 16kB | 1 | 4kB | 1 | 256 | 128 | 32 | 64 | 64 | 40 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1050** | 32kB | 1 | 4kB | 4 | 50 | 256 | 32 | 64 | 64 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1051** | 16kB | 4 | 32kB | 1 | 64 | 96 | 64 | 8 | 32 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1052** | 32kB | 4 | 64kB | 2 | 256 | 96 | 16 | 16 | 16 | 128 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1053** | 32kB | 4 | 32kB | 2 | 50 | 128 | 16 | 32 | 64 | 16 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1054** | 32kB | 2 | 16kB | 2 | 96 | 256 | 32 | 64 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1055** | 32kB | 2 | 4kB | 2 | 128 | 256 | 32 | 64 | 32 | 40 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1056** | 16kB | 2 | 8kB | 1 | 64 | 256 | 16 | 32 | 16 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | p. 143 |
| **#1057** | 64kB | 1 | 16kB | 1 | 64 | 128 | 64 | 8 | 32 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1058** | 32kB | 1 | 32kB | 4 | 50 | 128 | 64 | 16 | 16 | 64 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1059** | 16kB | 1 | 16kB | 4 | 128 | 256 | 16 | 16 | 64 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1060** | 16kB | 4 | 4kB | 4 | 96 | 256 | 64 | 16 | 32 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1061** | 16kB | 4 | 4kB | 2 | 256 | 128 | 16 | 64 | 8 | 16 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1062** | 32kB | 1 | 16kB | 1 | 64 | 96 | 16 | 64 | 64 | 64 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1063** | 16kB | 1 | 32kB | 1 | 64 | 128 | 32 | 32 | 32 | 128 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1064** | 64kB | 4 | 64kB | 4 | 128 | 96 | 32 | 64 | 8 | 128 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1065** | 32kB | 2 | 4kB | 4 | 128 | 128 | 64 | 8 | 64 | 16 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | p. 79 |
| **#1066** | 64kB | 2 | 4kB | 4 | 64 | 128 | 32 | 64 | 8 | 128 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1067** | 64kB | 2 | 8kB | 4 | 64 | 96 | 64 | 16 | 32 | 64 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1068** | 16kB | 2 | 64kB | 2 | 128 | 256 | 32 | 16 | 64 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1069** | 32kB | 4 | 8kB | 1 | 128 | 128 | 64 | 32 | 16 | 32 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1070** | 64kB | 2 | 64kB | 4 | 50 | 128 | 16 | 16 | 32 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1071** | 64kB | 2 | 8kB | 2 | 256 | 96 | 32 | 16 | 32 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1072** | 16kB | 2 | 32kB | 2 | 128 | 96 | 64 | 16 | 64 | 64 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1073** | 16kB | 2 | 16kB | 2 | 50 | 256 | 64 | 16 | 64 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1074** | 16kB | 4 | 4kB | 4 | 50 | 96 | 64 | 16 | 8 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1075** | 16kB | 2 | 64kB | 2 | 256 | 96 | 16 | 16 | 16 | 64 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1076** | 64kB | 1 | 16kB | 4 | 128 | 96 | 64 | 64 | 8 | 64 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1077** | 32kB | 4 | 64kB | 2 | 64 | 96 | 16 | 64 | 64 | 16 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1078** | 64kB | 4 | 32kB | 2 | 128 | 128 | 32 | 16 | 16 | 64 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1079** | 64kB | 2 | 16kB | 2 | 256 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1080** | 64kB | 4 | 32kB | 1 | 96 | 256 | 64 | 16 | 64 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1081** | 32kB | 2 | 4kB | 4 | 128 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1082** | 32kB | 1 | 8kB | 1 | 256 | 128 | 32 | 64 | 32 | 64 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1083** | 16kB | 4 | 8kB | 4 | 256 | 128 | 16 | 64 | 16 | 32 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1084** | 32kB | 4 | 8kB | 1 | 96 | 96 | 64 | 32 | 64 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1085** | 32kB | 1 | 8kB | 2 | 96 | 256 | 32 | 64 | 16 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1086** | 32kB | 4 | 8kB | 2 | 50 | 256 | 16 | 64 | 64 | 128 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1087** | 16kB | 2 | 32kB | 4 | 128 | 128 | 16 | 64 | 8 | 64 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1088** | 32kB | 2 | 32kB | 4 | 128 | 128 | 64 | 8 | 32 | 16 | 3 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1089** | 32kB | 1 | 4kB | 4 | 256 | 128 | 64 | 64 | 64 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1090** | 64kB | 2 | 16kB | 4 | 96 | 128 | 16 | 16 | 16 | 40 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1091** | 32kB | 4 | 64kB | 1 | 96 | 256 | 16 | 16 | 64 | 16 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1092** | 64kB | 4 | 32kB | 1 | 256 | 128 | 32 | 16 | 32 | 64 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1093** | 32kB | 2 | 4kB | 1 | 128 | 96 | 64 | 8 | 16 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1094** | 32kB | 2 | 32kB | 4 | 64 | 128 | 16 | 64 | 32 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1095** | 16kB | 4 | 16kB | 2 | 128 | 256 | 32 | 16 | 8 | 32 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1096** | 16kB | 4 | 32kB | 4 | 96 | 96 | 32 | 64 | 64 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1097** | 16kB | 1 | 4kB | 2 | 128 | 256 | 16 | 8 | 64 | 32 | 1 | $2^{13}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1098** | 32kB | 4 | 8kB | 4 | 96 | 128 | 64 | 16 | 8 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1099** | 64kB | 2 | 32kB | 4 | 64 | 128 | 64 | 8 | 8 | 128 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1100** | 64kB | 4 | 32kB | 2 | 64 | 128 | 64 | 16 | 64 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1101** | 64kB | 1 | 16kB | 1 | 128 | 96 | 32 | 8 | 16 | 64 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1102** | 32kB | 4 | 16kB | 2 | 256 | 256 | 64 | 16 | 8 | 16 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1103** | 64kB | 1 | 64kB | 1 | 96 | 128 | 32 | 16 | 8 | 64 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1104** | 64kB | 2 | 4kB | 1 | 128 | 96 | 16 | 16 | 32 | 16 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1105** | 32kB | 2 | 8kB | 1 | 96 | 256 | 32 | 64 | 8 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1106** | 32kB | 1 | 4kB | 4 | 50 | 256 | 32 | 64 | 32 | 16 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1107** | 16kB | 1 | 8kB | 2 | 256 | 96 | 64 | 64 | 64 | 16 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1108** | 64kB | 2 | 32kB | 1 | 64 | 128 | 32 | 64 | 64 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1109** | 32kB | 4 | 32kB | 4 | 256 | 128 | 16 | 32 | 32 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1110** | 16kB | 2 | 16kB | 1 | 96 | 96 | 64 | 8 | 8 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1111** | 16kB | 1 | 32kB | 2 | 256 | 96 | 32 | 8 | 16 | 128 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1112** | 64kB | 2 | 4kB | 1 | 256 | 96 | 16 | 32 | 64 | 64 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1113** | 64kB | 1 | 16kB | 1 | 256 | 96 | 32 | 32 | 64 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1114** | 64kB | 4 | 4kB | 4 | 64 | 96 | 64 | 16 | 16 | 32 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1115** | 64kB | 4 | 8kB | 1 | 50 | 256 | 64 | 8 | 32 | 32 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1116** | 32kB | 4 | 4kB | 2 | 50 | 96 | 16 | 8 | 16 | 128 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1117** | 32kB | 1 | 64kB | 4 | 96 | 96 | 64 | 64 | 32 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1118** | 64kB | 1 | 4kB | 1 | 50 | 96 | 32 | 16 | 16 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1119** | 32kB | 1 | 32kB | 4 | 96 | 256 | 32 | 8 | 16 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1120** | 16kB | 1 | 32kB | 1 | 64 | 96 | 16 | 64 | 32 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1121** | 64kB | 2 | 32kB | 1 | 256 | 256 | 64 | 64 | 8 | 32 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1122** | 16kB | 2 | 8kB | 1 | 256 | 128 | 64 | 64 | 32 | 32 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1123** | 32kB | 2 | 32kB | 2 | 128 | 96 | 32 | 8 | 32 | 40 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1124** | 16kB | 4 | 64kB | 4 | 64 | 128 | 32 | 8 | 8 | 16 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1125** | 16kB | 4 | 8kB | 2 | 256 | 128 | 32 | 8 | 64 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1126** | 16kB | 4 | 4kB | 1 | 50 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1127** | 16kB | 4 | 64kB | 2 | 96 | 256 | 64 | 8 | 32 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1128** | 64kB | 4 | 8kB | 2 | 64 | 96 | 16 | 8 | 64 | 16 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1129** | 32kB | 2 | 32kB | 2 | 128 | 256 | 32 | 8 | 8 | 40 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1130** | 32kB | 1 | 8kB | 4 | 256 | 256 | 32 | 64 | 16 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1131** | 64kB | 1 | 64kB | 2 | 128 | 256 | 32 | 32 | 64 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1132** | 16kB | 2 | 64kB | 2 | 96 | 128 | 32 | 16 | 32 | 64 | 3 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1133** | 16kB | 1 | 32kB | 1 | 256 | 256 | 16 | 16 | 64 | 64 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1134** | 64kB | 1 | 8kB | 1 | 256 | 128 | 32 | 8 | 32 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1135** | 16kB | 1 | 32kB | 1 | 64 | 256 | 32 | 64 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1136** | 64kB | 2 | 8kB | 1 | 96 | 128 | 16 | 32 | 16 | 40 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1137** | 64kB | 4 | 8kB | 1 | 96 | 128 | 16 | 32 | 32 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1138** | 16kB | 2 | 32kB | 1 | 50 | 96 | 16 | 64 | 8 | 64 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1139** | 32kB | 4 | 16kB | 2 | 96 | 128 | 32 | 32 | 8 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1140** | 64kB | 4 | 64kB | 4 | 64 | 128 | 16 | 16 | 16 | 128 | 3 | $2^{13}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1141** | 32kB | 2 | 4kB | 1 | 96 | 96 | 64 | 64 | 8 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1142** | 32kB | 4 | 16kB | 1 | 128 | 96 | 64 | 8 | 8 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1143** | 32kB | 1 | 64kB | 2 | 128 | 256 | 32 | 16 | 32 | 64 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1144** | 16kB | 1 | 8kB | 2 | 96 | 128 | 64 | 64 | 8 | 32 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1145** | 32kB | 1 | 4kB | 1 | 64 | 96 | 64 | 64 | 64 | 64 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1146** | 16kB | 4 | 16kB | 1 | 128 | 128 | 16 | 64 | 16 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1147** | 64kB | 4 | 16kB | 1 | 96 | 128 | 16 | 16 | 64 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1148** | 16kB | 4 | 32kB | 4 | 256 | 256 | 64 | 16 | 64 | 32 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1149** | 16kB | 4 | 64kB | 2 | 96 | 128 | 64 | 32 | 16 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1150** | 16kB | 4 | 4kB | 2 | 256 | 128 | 32 | 32 | 8 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1151** | 64kB | 4 | 32kB | 4 | 64 | 128 | 16 | 16 | 32 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1152** | 32kB | 4 | 8kB | 2 | 256 | 256 | 32 | 16 | 64 | 64 | 1 | $2^{11}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1153** | 16kB | 4 | 8kB | 4 | 64 | 96 | 64 | 8 | 64 | 128 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1154** | 32kB | 1 | 8kB | 4 | 50 | 96 | 32 | 32 | 64 | 64 | 1 | $2^{12}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1155** | 16kB | 2 | 4kB | 4 | 64 | 256 | 16 | 64 | 32 | 32 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1156** | 32kB | 4 | 4kB | 1 | 50 | 128 | 64 | 64 | 32 | 40 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1157** | 32kB | 4 | 8kB | 2 | 256 | 96 | 64 | 8 | 64 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1158** | 16kB | 1 | 32kB | 1 | 64 | 96 | 16 | 64 | 64 | 64 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1159** | 16kB | 2 | 8kB | 4 | 64 | 256 | 32 | 16 | 16 | 40 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1160** | 32kB | 2 | 8kB | 4 | 128 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1161** | 32kB | 4 | 8kB | 4 | 128 | 96 | 16 | 64 | 32 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1162** | 64kB | 4 | 8kB | 4 | 50 | 256 | 32 | 8 | 64 | 16 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1163** | 32kB | 1 | 8kB | 1 | 64 | 96 | 32 | 8 | 64 | 40 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1164** | 16kB | 4 | 8kB | 1 | 96 | 128 | 32 | 32 | 32 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1165** | 16kB | 4 | 4kB | 1 | 256 | 128 | 64 | 64 | 64 | 128 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | p. 142 |
| **#1166** | 32kB | 1 | 16kB | 4 | 64 | 96 | 32 | 8 | 16 | 16 | 3 | $2^{10}$ | 3 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1167** | 64kB | 4 | 4kB | 4 | 50 | 128 | 32 | 16 | 64 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1168** | 16kB | 4 | 64kB | 1 | 128 | 256 | 32 | 32 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1169** | 64kB | 4 | 32kB | 4 | 50 | 256 | 64 | 32 | 64 | 128 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1170** | 32kB | 1 | 8kB | 2 | 64 | 96 | 64 | 32 | 32 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1171** | 16kB | 4 | 8kB | 4 | 64 | 128 | 16 | 64 | 64 | 40 | 3 | $2^{11}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1172** | 16kB | 4 | 8kB | 4 | 64 | 128 | 16 | 32 | 64 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1173** | 64kB | 1 | 8kB | 1 | 96 | 256 | 32 | 32 | 16 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1174** | 64kB | 2 | 64kB | 2 | 50 | 96 | 64 | 32 | 8 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1175** | 64kB | 2 | 64kB | 2 | 50 | 96 | 16 | 32 | 16 | 64 | 3 | $2^{13}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1176** | 16kB | 4 | 4kB | 4 | 50 | 128 | 32 | 16 | 32 | 32 | 2 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1177** | 64kB | 4 | 4kB | 1 | 256 | 128 | 32 | 8 | 16 | 32 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1178** | 32kB | 2 | 16kB | 1 | 96 | 256 | 64 | 64 | 32 | 32 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1179** | 64kB | 4 | 16kB | 1 | 128 | 256 | 16 | 32 | 64 | 40 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1180** | 16kB | 4 | 16kB | 4 | 128 | 96 | 32 | 32 | 8 | 16 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1181** | 32kB | 1 | 4kB | 1 | 64 | 128 | 16 | 16 | 8 | 40 | 1 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | p. 142 |
| **#1182** | 64kB | 1 | 16kB | 4 | 50 | 256 | 64 | 32 | 16 | 32 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1183** | 16kB | 4 | 32kB | 2 | 128 | 256 | 64 | 16 | 16 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1184** | 64kB | 4 | 8kB | 1 | 64 | 96 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1185** | 64kB | 2 | 4kB | 4 | 96 | 128 | 16 | 32 | 64 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1186** | 64kB | 2 | 32kB | 4 | 96 | 96 | 16 | 16 | 32 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1187** | 32kB | 1 | 8kB | 4 | 128 | 128 | 16 | 32 | 32 | 128 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1188** | 16kB | 2 | 16kB | 4 | 96 | 96 | 16 | 64 | 64 | 64 | 3 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | p. 81 |
| **#1189** | 32kB | 2 | 16kB | 1 | 128 | 256 | 64 | 64 | 64 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1190** | 16kB | 4 | 16kB | 1 | 256 | 256 | 16 | 64 | 16 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1191** | 32kB | 4 | 4kB | 1 | 256 | 96 | 32 | 32 | 16 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1192** | 16kB | 2 | 32kB | 2 | 64 | 256 | 16 | 64 | 32 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1193** | 16kB | 4 | 32kB | 4 | 96 | 96 | 16 | 64 | 64 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1194** | 32kB | 1 | 64kB | 1 | 64 | 96 | 16 | 8 | 8 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1195** | 64kB | 1 | 8kB | 4 | 256 | 96 | 16 | 8 | 8 | 40 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1196** | 64kB | 2 | 4kB | 1 | 50 | 256 | 32 | 8 | 16 | 64 | 3 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1197** | 16kB | 2 | 32kB | 4 | 128 | 128 | 64 | 8 | 16 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1198** | 16kB | 1 | 16kB | 2 | 256 | 256 | 16 | 8 | 32 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1199** | 32kB | 4 | 8kB | 1 | 128 | 256 | 32 | 16 | 16 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1200** | 16kB | 4 | 16kB | 4 | 64 | 256 | 32 | 8 | 8 | 16 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1201** | 16kB | 1 | 4kB | 1 | 96 | 128 | 64 | 16 | 64 | 64 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1202** | 32kB | 4 | 64kB | 2 | 96 | 256 | 16 | 16 | 64 | 128 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1203** | 16kB | 2 | 8kB | 1 | 96 | 256 | 16 | 32 | 8 | 64 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1204** | 32kB | 1 | 4kB | 1 | 256 | 96 | 32 | 64 | 32 | 128 | 1 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1205** | 64kB | 4 | 8kB | 2 | 50 | 96 | 32 | 32 | 16 | 128 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1206** | 64kB | 2 | 4kB | 4 | 256 | 128 | 32 | 64 | 32 | 128 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1207** | 64kB | 4 | 8kB | 1 | 50 | 128 | 16 | 8 | 32 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1208** | 32kB | 1 | 8kB | 4 | 128 | 96 | 64 | 8 | 32 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1209** | 16kB | 2 | 32kB | 2 | 96 | 128 | 32 | 8 | 8 | 128 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1210** | 64kB | 1 | 8kB | 4 | 96 | 128 | 32 | 64 | 8 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1211** | 32kB | 4 | 32kB | 2 | 64 | 128 | 64 | 16 | 64 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1212** | 16kB | 4 | 8kB | 2 | 128 | 96 | 64 | 8 | 8 | 128 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1213** | 32kB | 4 | 16kB | 2 | 96 | 256 | 32 | 64 | 16 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1214** | 32kB | 4 | 16kB | 1 | 64 | 96 | 32 | 16 | 32 | 64 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1215** | 64kB | 2 | 4kB | 2 | 50 | 128 | 16 | 16 | 16 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1216** | 64kB | 4 | 64kB | 2 | 128 | 256 | 64 | 32 | 32 | 16 | 1 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1217** | 16kB | 2 | 32kB | 2 | 64 | 256 | 32 | 16 | 32 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1218** | 16kB | 4 | 8kB | 2 | 50 | 128 | 16 | 16 | 64 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1219** | 64kB | 1 | 16kB | 2 | 256 | 96 | 32 | 16 | 16 | 16 | 3 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1220** | 64kB | 4 | 64kB | 1 | 50 | 128 | 32 | 32 | 32 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1221** | 16kB | 1 | 4kB | 4 | 128 | 128 | 16 | 16 | 8 | 128 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1222** | 32kB | 4 | 8kB | 1 | 96 | 256 | 64 | 32 | 8 | 40 | 1 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1223** | 32kB | 1 | 32kB | 1 | 96 | 256 | 16 | 64 | 8 | 128 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1224** | 64kB | 1 | 32kB | 1 | 50 | 256 | 16 | 32 | 64 | 32 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1225** | 32kB | 2 | 64kB | 1 | 256 | 128 | 32 | 8 | 16 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1226** | 32kB | 2 | 4kB | 4 | 128 | 256 | 32 | 16 | 8 | 64 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1227** | 64kB | 1 | 4kB | 2 | 256 | 96 | 16 | 8 | 64 | 32 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1228** | 64kB | 1 | 32kB | 4 | 50 | 96 | 32 | 64 | 32 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1229** | 16kB | 1 | 64kB | 1 | 128 | 128 | 16 | 32 | 64 | 32 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1230** | 32kB | 1 | 16kB | 2 | 64 | 256 | 16 | 64 | 16 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1231** | 32kB | 4 | 4kB | 1 | 128 | 128 | 64 | 32 | 16 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1232** | 32kB | 2 | 32kB | 4 | 50 | 96 | 64 | 32 | 16 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1233** | 64kB | 1 | 4kB | 1 | 128 | 128 | 16 | 32 | 32 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1234** | 16kB | 4 | 16kB | 2 | 50 | 256 | 16 | 32 | 32 | 32 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1235** | 16kB | 2 | 4kB | 2 | 50 | 128 | 16 | 32 | 8 | 128 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1236** | 16kB | 4 | 32kB | 4 | 64 | 256 | 32 | 64 | 32 | 32 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1237** | 32kB | 2 | 64kB | 4 | 128 | 128 | 64 | 16 | 64 | 16 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1238** | 16kB | 1 | 32kB | 2 | 64 | 128 | 64 | 32 | 32 | 16 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1239** | 16kB | 4 | 16kB | 2 | 256 | 128 | 32 | 64 | 8 | 40 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1240** | 64kB | 1 | 32kB | 1 | 50 | 256 | 64 | 8 | 64 | 64 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1241** | 32kB | 2 | 16kB | 1 | 256 | 256 | 32 | 32 | 64 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1242** | 64kB | 4 | 64kB | 4 | 128 | 256 | 64 | 64 | 8 | 64 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1243** | 32kB | 4 | 32kB | 1 | 128 | 128 | 16 | 64 | 64 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1244** | 16kB | 1 | 8kB | 1 | 128 | 256 | 64 | 8 | 64 | 32 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1245** | 16kB | 4 | 64kB | 4 | 50 | 256 | 64 | 64 | 16 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1246** | 64kB | 4 | 32kB | 1 | 96 | 256 | 16 | 64 | 16 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1247** | 32kB | 4 | 8kB | 1 | 128 | 256 | 32 | 16 | 32 | 32 | 2 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1248** | 16kB | 2 | 8kB | 4 | 256 | 256 | 32 | 64 | 32 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1249** | 32kB | 1 | 16kB | 2 | 96 | 128 | 16 | 8 | 16 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1250** | 32kB | 1 | 64kB | 1 | 50 | 256 | 64 | 32 | 32 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1251** | 64kB | 1 | 4kB | 4 | 128 | 96 | 64 | 64 | 64 | 128 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1252** | 16kB | 1 | 8kB | 1 | 128 | 128 | 64 | 64 | 8 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1253** | 32kB | 2 | 4kB | 1 | 256 | 256 | 64 | 16 | 16 | 16 | 3 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1254** | 64kB | 4 | 8kB | 1 | 96 | 128 | 64 | 32 | 16 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1255** | 64kB | 2 | 64kB | 1 | 256 | 96 | 16 | 8 | 64 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1256** | 16kB | 4 | 32kB | 2 | 128 | 96 | 16 | 8 | 64 | 128 | 1 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1257** | 32kB | 1 | 64kB | 4 | 128 | 96 | 32 | 8 | 8 | 64 | 1 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1258** | 64kB | 1 | 4kB | 4 | 256 | 256 | 16 | 16 | 64 | 64 | 1 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1259** | 16kB | 1 | 8kB | 1 | 128 | 256 | 64 | 8 | 8 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1260** | 16kB | 2 | 32kB | 2 | 50 | 256 | 32 | 8 | 8 | 16 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1261** | 32kB | 4 | 4kB | 2 | 96 | 256 | 64 | 16 | 64 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1262** | 32kB | 2 | 64kB | 4 | 96 | 256 | 16 | 16 | 32 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1263** | 16kB | 1 | 8kB | 2 | 96 | 256 | 64 | 32 | 16 | 16 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1264** | 32kB | 1 | 16kB | 4 | 256 | 96 | 16 | 16 | 16 | 64 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1265** | 16kB | 4 | 32kB | 2 | 128 | 96 | 64 | 8 | 8 | 32 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1266** | 16kB | 4 | 16kB | 4 | 64 | 256 | 16 | 8 | 32 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1267** | 32kB | 2 | 8kB | 4 | 128 | 256 | 16 | 32 | 64 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1268** | 64kB | 4 | 4kB | 4 | 256 | 256 | 32 | 64 | 16 | 40 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1269** | 64kB | 2 | 8kB | 4 | 64 | 96 | 64 | 16 | 64 | 16 | 2 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1270** | 16kB | 1 | 32kB | 1 | 50 | 256 | 64 | 64 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1271** | 16kB | 1 | 32kB | 2 | 64 | 128 | 64 | 16 | 32 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1272** | 64kB | 4 | 64kB | 2 | 64 | 96 | 64 | 8 | 8 | 32 | 2 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1273** | 16kB | 1 | 4kB | 1 | 256 | 128 | 64 | 8 | 8 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1274** | 64kB | 4 | 64kB | 1 | 64 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1275** | 16kB | 4 | 32kB | 1 | 128 | 256 | 16 | 16 | 64 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1276** | 64kB | 4 | 4kB | 4 | 50 | 96 | 64 | 64 | 8 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1277** | 32kB | 4 | 8kB | 1 | 64 | 96 | 32 | 64 | 64 | 32 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1278** | 64kB | 1 | 32kB | 2 | 128 | 128 | 32 | 8 | 8 | 32 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1279** | 32kB | 2 | 64kB | 4 | 256 | 128 | 32 | 32 | 16 | 128 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1280** | 16kB | 1 | 16kB | 4 | 64 | 256 | 16 | 32 | 32 | 64 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1281** | 64kB | 1 | 4kB | 2 | 256 | 256 | 32 | 32 | 8 | 32 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1282** | 32kB | 2 | 4kB | 1 | 128 | 96 | 16 | 32 | 8 | 40 | 2 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1283** | 16kB | 2 | 4kB | 4 | 96 | 96 | 32 | 32 | 8 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1284** | 16kB | 2 | 64kB | 4 | 96 | 96 | 64 | 64 | 8 | 16 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1285** | 64kB | 4 | 64kB | 4 | 128 | 96 | 64 | 16 | 8 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1286** | 32kB | 2 | 16kB | 1 | 64 | 256 | 16 | 16 | 64 | 16 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1287** | 16kB | 1 | 8kB | 1 | 256 | 128 | 16 | 16 | 64 | 64 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1288** | 16kB | 1 | 4kB | 4 | 64 | 96 | 32 | 8 | 8 | 16 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | p. 142 |
| **#1289** | 64kB | 1 | 32kB | 4 | 50 | 128 | 64 | 32 | 64 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1290** | 32kB | 1 | 8kB | 2 | 128 | 128 | 16 | 16 | 64 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1291** | 64kB | 2 | 8kB | 1 | 64 | 96 | 64 | 32 | 64 | 40 | 2 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1292** | 32kB | 2 | 4kB | 2 | 96 | 256 | 32 | 16 | 16 | 40 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1293** | 32kB | 1 | 4kB | 2 | 96 | 256 | 32 | 8 | 8 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1294** | 64kB | 2 | 64kB | 4 | 128 | 128 | 32 | 32 | 16 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1295** | 32kB | 2 | 64kB | 4 | 64 | 128 | 16 | 32 | 32 | 16 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1296** | 32kB | 1 | 8kB | 1 | 128 | 128 | 16 | 32 | 16 | 32 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1297** | 16kB | 2 | 32kB | 2 | 96 | 96 | 16 | 32 | 32 | 40 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | p. 143 |
| **#1298** | 32kB | 4 | 64kB | 2 | 96 | 256 | 16 | 64 | 8 | 64 | 3 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1299** | 64kB | 2 | 8kB | 4 | 50 | 96 | 16 | 16 | 64 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1300** | 32kB | 1 | 16kB | 1 | 64 | 96 | 32 | 8 | 32 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1301** | 16kB | 1 | 64kB | 1 | 128 | 256 | 64 | 64 | 8 | 32 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1302** | 64kB | 4 | 64kB | 4 | 50 | 96 | 16 | 8 | 8 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1303** | 16kB | 2 | 4kB | 2 | 256 | 256 | 32 | 64 | 32 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1304** | 32kB | 4 | 16kB | 2 | 256 | 128 | 32 | 8 | 32 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1305** | 16kB | 4 | 64kB | 2 | 50 | 128 | 32 | 32 | 8 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1306** | 16kB | 2 | 4kB | 2 | 128 | 128 | 64 | 8 | 8 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1307** | 16kB | 4 | 4kB | 4 | 64 | 96 | 64 | 32 | 64 | 16 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1308** | 32kB | 2 | 8kB | 4 | 96 | 128 | 64 | 16 | 32 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1309** | 32kB | 1 | 8kB | 1 | 96 | 256 | 16 | 16 | 8 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1310** | 64kB | 4 | 4kB | 1 | 96 | 128 | 16 | 16 | 16 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1311** | 64kB | 2 | 4kB | 4 | 64 | 128 | 32 | 16 | 64 | 40 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1312** | 16kB | 1 | 4kB | 2 | 256 | 256 | 64 | 16 | 64 | 40 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1313** | 16kB | 2 | 8kB | 2 | 96 | 256 | 64 | 64 | 32 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1314** | 32kB | 1 | 8kB | 4 | 50 | 128 | 32 | 32 | 16 | 40 | 1 | $2^{14}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1315** | 16kB | 4 | 32kB | 2 | 64 | 96 | 64 | 8 | 8 | 64 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1316** | 32kB | 2 | 32kB | 1 | 128 | 256 | 32 | 16 | 16 | 16 | 3 | $2^{11}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1317** | 64kB | 1 | 16kB | 2 | 64 | 256 | 16 | 64 | 64 | 16 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1318** | 16kB | 1 | 16kB | 2 | 128 | 256 | 32 | 64 | 8 | 32 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1319** | 32kB | 4 | 64kB | 4 | 128 | 128 | 32 | 16 | 32 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1320** | 32kB | 4 | 4kB | 1 | 128 | 128 | 32 | 64 | 16 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1321** | 32kB | 4 | 8kB | 4 | 128 | 256 | 64 | 32 | 16 | 64 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1322** | 32kB | 2 | 32kB | 4 | 96 | 96 | 64 | 16 | 8 | 16 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1323** | 64kB | 4 | 8kB | 4 | 96 | 128 | 16 | 8 | 64 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1324** | 16kB | 4 | 4kB | 2 | 256 | 96 | 16 | 8 | 8 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1325** | 64kB | 1 | 32kB | 2 | 96 | 128 | 64 | 64 | 64 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1326** | 16kB | 4 | 4kB | 1 | 128 | 256 | 16 | 32 | 8 | 16 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1327** | 16kB | 2 | 64kB | 4 | 128 | 128 | 32 | 32 | 64 | 32 | 2 | $2^{13}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1328** | 64kB | 2 | 32kB | 2 | 256 | 128 | 64 | 32 | 16 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1329** | 16kB | 4 | 64kB | 2 | 256 | 96 | 32 | 32 | 8 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1330** | 16kB | 1 | 8kB | 1 | 256 | 96 | 16 | 8 | 64 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1331** | 16kB | 4 | 64kB | 4 | 64 | 96 | 64 | 8 | 8 | 40 | 2 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1332** | 64kB | 1 | 32kB | 1 | 64 | 128 | 64 | 8 | 64 | 40 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1333** | 32kB | 2 | 8kB | 1 | 50 | 96 | 16 | 16 | 8 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1334** | 64kB | 2 | 32kB | 4 | 64 | 96 | 32 | 32 | 16 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1335** | 32kB | 2 | 32kB | 1 | 50 | 96 | 64 | 16 | 32 | 40 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1336** | 32kB | 4 | 16kB | 4 | 64 | 256 | 16 | 64 | 8 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1337** | 64kB | 4 | 64kB | 4 | 96 | 128 | 16 | 8 | 32 | 16 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1338** | 16kB | 4 | 16kB | 1 | 96 | 96 | 32 | 8 | 8 | 16 | 1 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1339** | 16kB | 2 | 32kB | 4 | 96 | 128 | 16 | 64 | 16 | 128 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | p. 134 |
| **#1340** | 16kB | 4 | 16kB | 2 | 96 | 256 | 64 | 16 | 16 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1341** | 32kB | 1 | 4kB | 4 | 96 | 96 | 32 | 32 | 32 | 16 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1342** | 16kB | 1 | 64kB | 2 | 50 | 128 | 32 | 64 | 16 | 64 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1343** | 32kB | 4 | 32kB | 2 | 96 | 256 | 64 | 64 | 64 | 32 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1344** | 16kB | 1 | 16kB | 4 | 96 | 128 | 16 | 32 | 8 | 16 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1345** | 16kB | 2 | 64kB | 1 | 96 | 96 | 16 | 64 | 8 | 16 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1346** | 16kB | 1 | 8kB | 4 | 64 | 128 | 32 | 64 | 8 | 128 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1347** | 64kB | 2 | 4kB | 1 | 64 | 96 | 64 | 32 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1348** | 16kB | 1 | 16kB | 4 | 50 | 128 | 32 | 8 | 32 | 16 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1349** | 32kB | 1 | 4kB | 2 | 50 | 128 | 16 | 8 | 16 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1350** | 32kB | 4 | 4kB | 2 | 256 | 256 | 16 | 32 | 8 | 32 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1351** | 64kB | 2 | 4kB | 2 | 50 | 96 | 32 | 64 | 32 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1352** | 32kB | 4 | 4kB | 4 | 64 | 256 | 32 | 8 | 16 | 40 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1353** | 32kB | 1 | 4kB | 1 | 128 | 256 | 16 | 8 | 8 | 32 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1354** | 64kB | 2 | 32kB | 1 | 128 | 128 | 32 | 8 | 8 | 16 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1355** | 16kB | 2 | 8kB | 4 | 50 | 256 | 16 | 16 | 16 | 16 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1356** | 64kB | 1 | 16kB | 1 | 64 | 96 | 16 | 16 | 32 | 16 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1357** | 64kB | 1 | 32kB | 4 | 50 | 256 | 16 | 16 | 16 | 40 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1358** | 32kB | 4 | 32kB | 4 | 128 | 96 | 16 | 32 | 32 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1359** | 32kB | 1 | 64kB | 2 | 50 | 128 | 32 | 16 | 64 | 40 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1360** | 32kB | 1 | 4kB | 2 | 128 | 96 | 32 | 32 | 64 | 16 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1361** | 64kB | 4 | 4kB | 1 | 64 | 256 | 64 | 64 | 32 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1362** | 32kB | 1 | 64kB | 2 | 128 | 128 | 32 | 64 | 8 | 128 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1363** | 64kB | 4 | 32kB | 1 | 256 | 256 | 16 | 64 | 32 | 64 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1364** | 16kB | 4 | 32kB | 1 | 50 | 256 | 32 | 8 | 16 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1365** | 64kB | 2 | 32kB | 4 | 256 | 96 | 64 | 8 | 64 | 40 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1366** | 16kB | 4 | 8kB | 4 | 128 | 96 | 32 | 32 | 16 | 64 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1367** | 64kB | 2 | 32kB | 1 | 128 | 96 | 32 | 8 | 32 | 16 | 2 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1368** | 32kB | 4 | 32kB | 4 | 96 | 128 | 64 | 8 | 32 | 128 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1369** | 64kB | 1 | 16kB | 1 | 128 | 96 | 64 | 8 | 64 | 40 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1370** | 32kB | 4 | 4kB | 2 | 96 | 256 | 16 | 8 | 64 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1371** | 32kB | 1 | 8kB | 4 | 256 | 256 | 16 | 32 | 8 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1372** | 16kB | 2 | 16kB | 1 | 256 | 256 | 16 | 8 | 16 | 64 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1373** | 16kB | 1 | 64kB | 4 | 96 | 96 | 16 | 64 | 32 | 64 | 1 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1374** | 64kB | 1 | 4kB | 2 | 96 | 96 | 64 | 64 | 64 | 16 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1375** | 16kB | 1 | 64kB | 4 | 64 | 256 | 32 | 64 | 64 | 64 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1376** | 64kB | 2 | 16kB | 1 | 128 | 96 | 64 | 64 | 64 | 32 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1377** | 64kB | 4 | 4kB | 2 | 96 | 128 | 32 | 8 | 64 | 128 | 3 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1378** | 64kB | 1 | 32kB | 1 | 50 | 256 | 64 | 16 | 32 | 32 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1379** | 32kB | 4 | 4kB | 2 | 50 | 96 | 16 | 32 | 32 | 40 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1380** | 64kB | 1 | 16kB | 4 | 128 | 96 | 16 | 8 | 64 | 40 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1381** | 32kB | 1 | 16kB | 2 | 128 | 128 | 64 | 16 | 16 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1382** | 64kB | 2 | 32kB | 1 | 256 | 96 | 16 | 32 | 64 | 128 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1383** | 64kB | 2 | 64kB | 2 | 128 | 128 | 16 | 8 | 16 | 16 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1384** | 32kB | 1 | 8kB | 2 | 64 | 128 | 64 | 16 | 8 | 128 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1385** | 16kB | 2 | 32kB | 2 | 64 | 256 | 32 | 8 | 32 | 64 | 1 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1386** | 32kB | 1 | 32kB | 2 | 50 | 128 | 64 | 32 | 8 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1387** | 64kB | 4 | 4kB | 4 | 96 | 96 | 32 | 64 | 8 | 40 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1388** | 64kB | 4 | 16kB | 2 | 64 | 128 | 32 | 64 | 8 | 16 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1389** | 32kB | 4 | 8kB | 1 | 50 | 96 | 32 | 8 | 64 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1390** | 16kB | 4 | 32kB | 4 | 96 | 96 | 16 | 32 | 64 | 32 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1391** | 64kB | 4 | 8kB | 2 | 64 | 256 | 64 | 64 | 8 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1392** | 32kB | 2 | 32kB | 1 | 50 | 96 | 64 | 64 | 16 | 64 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1393** | 32kB | 2 | 32kB | 2 | 96 | 128 | 64 | 64 | 8 | 32 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1394** | 64kB | 4 | 64kB | 4 | 50 | 128 | 32 | 32 | 8 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1395** | 64kB | 2 | 16kB | 1 | 128 | 96 | 16 | 64 | 8 | 128 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1396** | 64kB | 4 | 32kB | 1 | 50 | 128 | 64 | 8 | 16 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1397** | 16kB | 2 | 64kB | 4 | 96 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1398** | 32kB | 2 | 4kB | 1 | 96 | 128 | 16 | 8 | 64 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1399** | 16kB | 1 | 64kB | 2 | 64 | 256 | 64 | 32 | 64 | 40 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1400** | 64kB | 2 | 64kB | 4 | 50 | 128 | 64 | 8 | 64 | 40 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1401** | 64kB | 2 | 8kB | 4 | 64 | 96 | 16 | 32 | 32 | 128 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1402** | 32kB | 4 | 32kB | 1 | 256 | 128 | 64 | 32 | 32 | 32 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1403** | 64kB | 2 | 4kB | 1 | 256 | 96 | 32 | 8 | 16 | 128 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1404** | 16kB | 2 | 16kB | 4 | 256 | 96 | 16 | 32 | 8 | 128 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1405** | 64kB | 2 | 8kB | 2 | 256 | 128 | 16 | 32 | 32 | 40 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1406** | 16kB | 1 | 32kB | 4 | 64 | 128 | 32 | 64 | 32 | 128 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1407** | 64kB | 2 | 16kB | 2 | 50 | 128 | 16 | 16 | 64 | 16 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1408** | 16kB | 4 | 32kB | 2 | 96 | 256 | 64 | 64 | 8 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1409** | 64kB | 4 | 8kB | 2 | 128 | 128 | 16 | 8 | 32 | 32 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1410** | 16kB | 2 | 4kB | 2 | 50 | 256 | 32 | 16 | 8 | 32 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1411** | 32kB | 1 | 64kB | 4 | 256 | 256 | 16 | 8 | 32 | 40 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1412** | 16kB | 1 | 4kB | 4 | 50 | 256 | 16 | 32 | 8 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1413** | 16kB | 4 | 4kB | 1 | 128 | 96 | 64 | 32 | 64 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1414** | 64kB | 1 | 16kB | 2 | 256 | 96 | 32 | 64 | 8 | 16 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1415** | 32kB | 1 | 16kB | 2 | 128 | 128 | 16 | 8 | 64 | 32 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1416** | 64kB | 1 | 16kB | 1 | 96 | 128 | 32 | 8 | 16 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1417** | 32kB | 4 | 4kB | 4 | 64 | 128 | 16 | 16 | 16 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1418** | 16kB | 4 | 64kB | 2 | 96 | 256 | 64 | 64 | 8 | 16 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1419** | 16kB | 2 | 8kB | 4 | 50 | 128 | 32 | 64 | 16 | 128 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1420** | 32kB | 2 | 64kB | 1 | 128 | 256 | 16 | 16 | 8 | 128 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1421** | 64kB | 1 | 8kB | 1 | 50 | 128 | 16 | 16 | 16 | 40 | 3 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1422** | 16kB | 1 | 8kB | 1 | 50 | 96 | 16 | 8 | 16 | 40 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1423** | 32kB | 4 | 8kB | 1 | 50 | 96 | 32 | 16 | 64 | 128 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1424** | 32kB | 1 | 32kB | 1 | 128 | 128 | 64 | 8 | 16 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1425** | 16kB | 4 | 64kB | 4 | 128 | 128 | 32 | 64 | 64 | 16 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1426** | 16kB | 1 | 4kB | 4 | 96 | 256 | 64 | 32 | 32 | 16 | 1 | $2^{11}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1427** | 32kB | 4 | 16kB | 4 | 128 | 96 | 64 | 8 | 32 | 16 | 3 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1428** | 16kB | 2 | 64kB | 4 | 256 | 96 | 16 | 8 | 16 | 16 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1429** | 32kB | 1 | 16kB | 2 | 50 | 96 | 16 | 32 | 16 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1430** | 64kB | 2 | 16kB | 1 | 256 | 96 | 64 | 16 | 8 | 64 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1431** | 64kB | 1 | 64kB | 4 | 256 | 128 | 16 | 64 | 8 | 40 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1432** | 32kB | 2 | 32kB | 1 | 96 | 96 | 16 | 64 | 64 | 64 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1433** | 16kB | 2 | 64kB | 1 | 128 | 96 | 64 | 8 | 8 | 64 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1434** | 64kB | 2 | 64kB | 4 | 128 | 96 | 32 | 8 | 8 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1435** | 16kB | 1 | 64kB | 1 | 128 | 256 | 16 | 8 | 32 | 64 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1436** | 32kB | 4 | 4kB | 4 | 256 | 256 | 32 | 64 | 8 | 32 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1437** | 16kB | 4 | 8kB | 4 | 64 | 96 | 32 | 16 | 32 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1438** | 16kB | 2 | 8kB | 2 | 50 | 128 | 64 | 64 | 8 | 16 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1439** | 32kB | 2 | 16kB | 1 | 128 | 96 | 64 | 64 | 8 | 16 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1440** | 64kB | 1 | 4kB | 2 | 128 | 256 | 16 | 8 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1441** | 64kB | 2 | 64kB | 4 | 64 | 256 | 64 | 8 | 32 | 128 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1442** | 32kB | 1 | 4kB | 2 | 96 | 96 | 64 | 32 | 8 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1443** | 16kB | 4 | 4kB | 1 | 50 | 256 | 64 | 8 | 32 | 128 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1444** | 64kB | 1 | 32kB | 4 | 128 | 128 | 32 | 64 | 64 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1445** | 16kB | 1 | 64kB | 2 | 64 | 256 | 64 | 16 | 16 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1446** | 64kB | 4 | 8kB | 4 | 50 | 96 | 64 | 32 | 32 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1447** | 16kB | 4 | 4kB | 1 | 50 | 256 | 32 | 16 | 8 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1448** | 64kB | 1 | 4kB | 4 | 256 | 128 | 64 | 32 | 8 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1449** | 32kB | 2 | 64kB | 4 | 96 | 256 | 32 | 64 | 64 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | p. 142 |
| **#1450** | 16kB | 1 | 32kB | 2 | 128 | 256 | 16 | 16 | 64 | 16 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1451** | 16kB | 4 | 4kB | 2 | 256 | 256 | 64 | 8 | 32 | 16 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1452** | 16kB | 2 | 16kB | 4 | 64 | 256 | 16 | 32 | 16 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1453** | 16kB | 2 | 8kB | 4 | 96 | 256 | 64 | 16 | 8 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1454** | 64kB | 1 | 16kB | 1 | 256 | 256 | 64 | 8 | 32 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1455** | 64kB | 1 | 4kB | 2 | 128 | 128 | 16 | 8 | 64 | 32 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1456** | 16kB | 1 | 4kB | 2 | 128 | 96 | 64 | 64 | 32 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1457** | 64kB | 1 | 32kB | 2 | 128 | 96 | 32 | 32 | 32 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1458** | 16kB | 2 | 64kB | 4 | 96 | 96 | 32 | 16 | 32 | 64 | 2 | $2^{12}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1459** | 64kB | 2 | 64kB | 2 | 50 | 128 | 32 | 64 | 64 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1460** | 32kB | 4 | 32kB | 2 | 128 | 96 | 64 | 32 | 16 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1461** | 32kB | 1 | 16kB | 4 | 50 | 256 | 16 | 32 | 8 | 40 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1462** | 16kB | 2 | 64kB | 2 | 128 | 128 | 16 | 32 | 32 | 32 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1463** | 64kB | 1 | 16kB | 4 | 128 | 128 | 64 | 16 | 32 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1464** | 64kB | 1 | 32kB | 1 | 256 | 128 | 64 | 16 | 64 | 64 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1465** | 64kB | 4 | 32kB | 2 | 128 | 96 | 32 | 64 | 16 | 16 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1466** | 32kB | 1 | 4kB | 1 | 64 | 128 | 16 | 64 | 32 | 16 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1467** | 32kB | 2 | 64kB | 1 | 256 | 256 | 32 | 32 | 16 | 64 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1468** | 32kB | 4 | 16kB | 4 | 256 | 128 | 32 | 32 | 64 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1469** | 64kB | 4 | 64kB | 4 | 128 | 96 | 32 | 8 | 16 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1470** | 16kB | 2 | 4kB | 2 | 64 | 256 | 32 | 8 | 32 | 32 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1471** | 16kB | 2 | 4kB | 2 | 64 | 128 | 32 | 32 | 16 | 64 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1472** | 64kB | 2 | 64kB | 4 | 64 | 96 | 16 | 64 | 64 | 40 | 2 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1473** | 64kB | 2 | 64kB | 1 | 256 | 96 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1474** | 16kB | 2 | 4kB | 2 | 64 | 96 | 16 | 16 | 16 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1475** | 16kB | 2 | 64kB | 2 | 50 | 128 | 32 | 64 | 16 | 32 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1476** | 64kB | 2 | 64kB | 2 | 128 | 96 | 64 | 16 | 64 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1477** | 32kB | 4 | 8kB | 2 | 50 | 256 | 32 | 8 | 16 | 40 | 1 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1478** | 32kB | 1 | 16kB | 4 | 64 | 256 | 64 | 16 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1479** | 16kB | 4 | 64kB | 4 | 256 | 128 | 32 | 16 | 64 | 16 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1480** | 64kB | 2 | 16kB | 4 | 256 | 96 | 32 | 32 | 8 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1481** | 32kB | 2 | 32kB | 4 | 128 | 256 | 32 | 64 | 32 | 16 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1482** | 32kB | 4 | 64kB | 4 | 50 | 96 | 64 | 16 | 16 | 32 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1483** | 32kB | 2 | 16kB | 2 | 50 | 128 | 32 | 32 | 16 | 32 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1484** | 64kB | 4 | 8kB | 1 | 96 | 256 | 64 | 32 | 64 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1485** | 64kB | 2 | 4kB | 2 | 128 | 96 | 64 | 32 | 32 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1486** | 16kB | 4 | 8kB | 4 | 256 | 96 | 32 | 64 | 32 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1487** | 64kB | 2 | 32kB | 1 | 128 | 256 | 32 | 8 | 64 | 64 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1488** | 64kB | 1 | 64kB | 2 | 256 | 256 | 64 | 64 | 32 | 16 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1489** | 32kB | 1 | 16kB | 1 | 256 | 256 | 16 | 16 | 16 | 40 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1490** | 64kB | 1 | 4kB | 4 | 64 | 128 | 16 | 8 | 32 | 40 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1491** | 64kB | 4 | 64kB | 1 | 96 | 96 | 64 | 32 | 16 | 128 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1492** | 64kB | 4 | 16kB | 4 | 50 | 128 | 16 | 64 | 64 | 40 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1493** | 32kB | 4 | 64kB | 1 | 50 | 128 | 32 | 16 | 16 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1494** | 16kB | 2 | 64kB | 1 | 64 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1495** | 64kB | 1 | 8kB | 4 | 256 | 96 | 32 | 64 | 16 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1496** | 64kB | 1 | 4kB | 2 | 128 | 128 | 16 | 64 | 16 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1497** | 64kB | 1 | 32kB | 4 | 128 | 256 | 16 | 8 | 64 | 128 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1498** | 32kB | 4 | 4kB | 2 | 256 | 256 | 32 | 16 | 16 | 40 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1499** | 64kB | 1 | 16kB | 1 | 128 | 96 | 32 | 8 | 64 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1500** | 16kB | 4 | 8kB | 2 | 64 | 256 | 64 | 32 | 8 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1501** | 16kB | 1 | 64kB | 1 | 50 | 128 | 32 | 64 | 64 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1502** | 64kB | 4 | 4kB | 1 | 96 | 128 | 32 | 32 | 16 | 40 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1503** | 64kB | 4 | 32kB | 2 | 64 | 128 | 32 | 64 | 8 | 16 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1504** | 64kB | 2 | 16kB | 2 | 50 | 96 | 16 | 32 | 16 | 128 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1505** | 32kB | 1 | 16kB | 2 | 256 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1506** | 16kB | 2 | 16kB | 4 | 128 | 128 | 64 | 64 | 8 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1507** | 64kB | 1 | 64kB | 4 | 128 | 128 | 64 | 16 | 8 | 16 | 2 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1508** | 16kB | 1 | 4kB | 1 | 256 | 256 | 32 | 64 | 64 | 40 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1509** | 64kB | 2 | 4kB | 4 | 96 | 128 | 16 | 64 | 16 | 128 | 3 | $2^{10}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1510** | 64kB | 4 | 64kB | 2 | 128 | 96 | 32 | 32 | 32 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1511** | 32kB | 1 | 8kB | 4 | 128 | 256 | 64 | 32 | 32 | 32 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1512** | 64kB | 1 | 8kB | 2 | 256 | 128 | 64 | 16 | 16 | 64 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1513** | 32kB | 1 | 64kB | 4 | 128 | 128 | 16 | 8 | 16 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1514** | 32kB | 2 | 16kB | 4 | 96 | 96 | 32 | 8 | 16 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1515** | 64kB | 1 | 8kB | 4 | 256 | 128 | 16 | 32 | 16 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1516** | 32kB | 1 | 4kB | 4 | 128 | 96 | 64 | 64 | 64 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1517** | 16kB | 2 | 64kB | 1 | 64 | 256 | 16 | 8 | 16 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1518** | 64kB | 4 | 8kB | 1 | 256 | 256 | 32 | 16 | 8 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1519** | 64kB | 1 | 64kB | 1 | 256 | 128 | 64 | 64 | 16 | 16 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1520** | 32kB | 1 | 64kB | 1 | 96 | 256 | 16 | 64 | 16 | 64 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1521** | 16kB | 4 | 64kB | 4 | 64 | 256 | 16 | 16 | 32 | 128 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1522** | 16kB | 1 | 32kB | 1 | 64 | 128 | 32 | 8 | 64 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1523** | 64kB | 2 | 4kB | 4 | 256 | 256 | 32 | 8 | 8 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1524** | 32kB | 1 | 64kB | 2 | 50 | 128 | 64 | 16 | 64 | 40 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1525** | 16kB | 1 | 16kB | 1 | 256 | 96 | 64 | 32 | 8 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1526** | 32kB | 1 | 64kB | 4 | 128 | 256 | 64 | 32 | 16 | 40 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1527** | 64kB | 1 | 8kB | 2 | 256 | 256 | 64 | 16 | 32 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1528** | 32kB | 2 | 4kB | 1 | 96 | 256 | 32 | 32 | 32 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1529** | 64kB | 4 | 16kB | 4 | 64 | 128 | 32 | 16 | 16 | 16 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1530** | 16kB | 2 | 64kB | 2 | 64 | 256 | 32 | 64 | 8 | 40 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1531** | 16kB | 4 | 8kB | 1 | 50 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1532** | 16kB | 1 | 4kB | 2 | 128 | 96 | 16 | 8 | 64 | 16 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1533** | 64kB | 4 | 64kB | 4 | 96 | 96 | 64 | 32 | 32 | 16 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1534** | 32kB | 4 | 8kB | 2 | 64 | 96 | 16 | 16 | 32 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1535** | 64kB | 4 | 64kB | 2 | 128 | 128 | 16 | 64 | 64 | 16 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1536** | 32kB | 1 | 32kB | 1 | 256 | 128 | 64 | 32 | 64 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1537** | 32kB | 1 | 64kB | 2 | 96 | 128 | 64 | 8 | 64 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1538** | 32kB | 2 | 32kB | 2 | 96 | 256 | 64 | 16 | 16 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1539** | 32kB | 1 | 32kB | 1 | 96 | 128 | 64 | 8 | 8 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1540** | 64kB | 1 | 16kB | 4 | 96 | 256 | 32 | 8 | 16 | 64 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1541** | 64kB | 1 | 8kB | 4 | 128 | 96 | 64 | 16 | 16 | 40 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1542** | 64kB | 4 | 4kB | 2 | 50 | 128 | 16 | 16 | 8 | 40 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1543** | 16kB | 1 | 32kB | 1 | 256 | 256 | 16 | 16 | 32 | 32 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1544** | 64kB | 4 | 8kB | 2 | 256 | 96 | 64 | 8 | 8 | 16 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1545** | 64kB | 2 | 64kB | 4 | 64 | 256 | 64 | 32 | 64 | 40 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1546** | 16kB | 2 | 4kB | 1 | 64 | 96 | 32 | 8 | 8 | 16 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | p. 79 |
| **#1547** | 16kB | 4 | 64kB | 4 | 128 | 128 | 64 | 32 | 64 | 32 | 1 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1548** | 32kB | 4 | 8kB | 2 | 256 | 128 | 64 | 32 | 16 | 16 | 3 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1549** | 64kB | 2 | 8kB | 1 | 96 | 96 | 16 | 32 | 8 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1550** | 32kB | 2 | 32kB | 4 | 96 | 96 | 16 | 16 | 32 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1551** | 32kB | 2 | 8kB | 2 | 128 | 96 | 32 | 16 | 64 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1552** | 32kB | 1 | 8kB | 4 | 256 | 256 | 16 | 64 | 32 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1553** | 32kB | 1 | 4kB | 2 | 256 | 256 | 32 | 64 | 8 | 64 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1554** | 32kB | 2 | 4kB | 4 | 64 | 96 | 16 | 64 | 16 | 32 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1555** | 16kB | 2 | 64kB | 4 | 50 | 96 | 16 | 32 | 32 | 40 | 1 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1556** | 32kB | 4 | 8kB | 1 | 50 | 256 | 64 | 32 | 8 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1557** | 32kB | 4 | 8kB | 4 | 96 | 96 | 64 | 64 | 16 | 40 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1558** | 32kB | 1 | 64kB | 2 | 50 | 96 | 16 | 16 | 8 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1559** | 16kB | 2 | 8kB | 4 | 50 | 96 | 16 | 8 | 16 | 32 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1560** | 32kB | 1 | 8kB | 1 | 256 | 96 | 32 | 32 | 8 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1561** | 32kB | 4 | 4kB | 1 | 96 | 256 | 32 | 64 | 8 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1562** | 64kB | 4 | 8kB | 4 | 50 | 96 | 64 | 8 | 64 | 32 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1563** | 32kB | 2 | 8kB | 4 | 128 | 96 | 32 | 16 | 64 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1564** | 64kB | 2 | 32kB | 2 | 128 | 96 | 64 | 16 | 32 | 40 | 2 | $2^{12}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1565** | 32kB | 2 | 32kB | 1 | 64 | 96 | 16 | 8 | 16 | 40 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1566** | 16kB | 2 | 16kB | 2 | 96 | 256 | 16 | 16 | 16 | 40 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1567** | 32kB | 1 | 16kB | 2 | 64 | 256 | 64 | 8 | 16 | 64 | 1 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1568** | 64kB | 4 | 32kB | 2 | 128 | 256 | 64 | 8 | 16 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1569** | 16kB | 1 | 4kB | 1 | 64 | 256 | 32 | 8 | 16 | 64 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1570** | 32kB | 1 | 4kB | 1 | 64 | 256 | 32 | 16 | 16 | 128 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1571** | 64kB | 2 | 16kB | 2 | 128 | 128 | 32 | 32 | 32 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1572** | 16kB | 4 | 32kB | 4 | 256 | 96 | 64 | 32 | 64 | 16 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1573** | 32kB | 4 | 32kB | 4 | 64 | 96 | 32 | 8 | 16 | 128 | 1 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1574** | 16kB | 2 | 16kB | 4 | 128 | 96 | 64 | 8 | 64 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1575** | 64kB | 4 | 8kB | 1 | 128 | 128 | 32 | 8 | 32 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1576** | 16kB | 2 | 8kB | 1 | 64 | 128 | 16 | 8 | 32 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1577** | 64kB | 1 | 32kB | 2 | 64 | 128 | 64 | 8 | 16 | 40 | 2 | $2^{11}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1578** | 16kB | 1 | 64kB | 1 | 128 | 128 | 16 | 32 | 8 | 32 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1579** | 32kB | 2 | 64kB | 2 | 50 | 256 | 16 | 8 | 16 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1580** | 16kB | 4 | 16kB | 2 | 50 | 256 | 16 | 64 | 64 | 128 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1581** | 16kB | 4 | 64kB | 4 | 50 | 128 | 16 | 8 | 64 | 32 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1582** | 16kB | 1 | 8kB | 4 | 64 | 128 | 32 | 8 | 16 | 32 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1583** | 32kB | 4 | 16kB | 2 | 96 | 128 | 32 | 8 | 32 | 32 | 1 | $2^{14}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1584** | 16kB | 4 | 32kB | 1 | 96 | 256 | 32 | 8 | 8 | 32 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1585** | 64kB | 2 | 4kB | 2 | 256 | 128 | 64 | 16 | 32 | 128 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1586** | 16kB | 4 | 8kB | 2 | 128 | 96 | 64 | 16 | 32 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1587** | 32kB | 4 | 64kB | 2 | 128 | 96 | 64 | 8 | 8 | 16 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1588** | 32kB | 2 | 64kB | 2 | 50 | 96 | 64 | 64 | 32 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1589** | 16kB | 4 | 64kB | 1 | 96 | 256 | 16 | 64 | 32 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1590** | 64kB | 2 | 16kB | 1 | 50 | 128 | 32 | 32 | 8 | 128 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1591** | 32kB | 2 | 8kB | 1 | 256 | 96 | 64 | 32 | 64 | 16 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1592** | 32kB | 2 | 8kB | 2 | 50 | 96 | 16 | 8 | 8 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1593** | 32kB | 4 | 64kB | 4 | 50 | 256 | 64 | 8 | 8 | 32 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1594** | 32kB | 2 | 8kB | 2 | 64 | 96 | 64 | 64 | 32 | 32 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1595** | 64kB | 2 | 16kB | 4 | 64 | 256 | 64 | 16 | 16 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1596** | 16kB | 4 | 8kB | 2 | 128 | 96 | 16 | 32 | 64 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1597** | 32kB | 4 | 16kB | 2 | 128 | 96 | 32 | 8 | 64 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1598** | 32kB | 4 | 16kB | 2 | 50 | 128 | 16 | 8 | 16 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1599** | 64kB | 1 | 4kB | 1 | 128 | 256 | 16 | 32 | 32 | 128 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1600** | 16kB | 2 | 64kB | 4 | 64 | 128 | 16 | 32 | 32 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1601** | 32kB | 2 | 64kB | 1 | 64 | 96 | 32 | 16 | 32 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1602** | 16kB | 2 | 32kB | 1 | 64 | 128 | 64 | 16 | 8 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1603** | 16kB | 4 | 32kB | 1 | 50 | 96 | 64 | 32 | 16 | 32 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1604** | 16kB | 1 | 64kB | 4 | 50 | 128 | 64 | 32 | 8 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1605** | 16kB | 2 | 4kB | 1 | 64 | 256 | 16 | 16 | 64 | 40 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1606** | 32kB | 2 | 16kB | 4 | 128 | 96 | 32 | 32 | 32 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1607** | 64kB | 2 | 64kB | 4 | 96 | 256 | 16 | 32 | 64 | 128 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1608** | 32kB | 4 | 64kB | 2 | 128 | 96 | 64 | 64 | 16 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1609** | 16kB | 2 | 4kB | 4 | 128 | 256 | 64 | 16 | 64 | 16 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1610** | 64kB | 4 | 32kB | 4 | 96 | 128 | 32 | 16 | 32 | 16 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1611** | 16kB | 1 | 64kB | 4 | 64 | 128 | 16 | 64 | 32 | 128 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1612** | 16kB | 4 | 64kB | 2 | 256 | 128 | 64 | 8 | 64 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1613** | 32kB | 4 | 32kB | 4 | 50 | 256 | 16 | 16 | 32 | 40 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1614** | 64kB | 2 | 4kB | 2 | 256 | 256 | 32 | 8 | 64 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1615** | 64kB | 2 | 32kB | 2 | 64 | 256 | 32 | 8 | 64 | 32 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1616** | 32kB | 1 | 64kB | 1 | 64 | 256 | 16 | 8 | 8 | 32 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1617** | 64kB | 4 | 32kB | 2 | 50 | 128 | 32 | 8 | 8 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1618** | 32kB | 2 | 16kB | 4 | 96 | 256 | 16 | 32 | 8 | 64 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1619** | 64kB | 2 | 32kB | 4 | 256 | 256 | 64 | 8 | 8 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1620** | 64kB | 1 | 64kB | 1 | 96 | 96 | 64 | 64 | 8 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1621** | 32kB | 4 | 16kB | 2 | 96 | 256 | 32 | 16 | 32 | 128 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1622** | 16kB | 2 | 4kB | 2 | 50 | 256 | 64 | 8 | 16 | 40 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1623** | 64kB | 4 | 16kB | 4 | 96 | 96 | 64 | 16 | 64 | 32 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1624** | 32kB | 2 | 32kB | 1 | 64 | 96 | 64 | 64 | 8 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1625** | 64kB | 1 | 8kB | 4 | 50 | 96 | 16 | 64 | 16 | 128 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1626** | 32kB | 2 | 32kB | 4 | 96 | 96 | 16 | 32 | 32 | 40 | 1 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1627** | 64kB | 2 | 16kB | 1 | 64 | 128 | 32 | 8 | 8 | 32 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1628** | 16kB | 4 | 16kB | 4 | 96 | 96 | 32 | 64 | 8 | 40 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1629** | 16kB | 2 | 16kB | 4 | 50 | 128 | 64 | 16 | 16 | 128 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1630** | 32kB | 4 | 16kB | 4 | 64 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1631** | 16kB | 1 | 32kB | 4 | 96 | 128 | 16 | 64 | 32 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | p. 142 |
| **#1632** | 32kB | 4 | 32kB | 2 | 64 | 256 | 32 | 8 | 32 | 32 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#1633** | 16kB | 4 | 16kB | 2 | 128 | 256 | 32 | 16 | 16 | 128 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1634** | 32kB | 2 | 8kB | 2 | 96 | 96 | 64 | 8 | 16 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1635** | 16kB | 4 | 16kB | 4 | 256 | 256 | 16 | 32 | 64 | 32 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1636** | 32kB | 4 | 16kB | 4 | 96 | 256 | 32 | 8 | 16 | 64 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1637** | 64kB | 2 | 16kB | 4 | 96 | 256 | 16 | 8 | 64 | 64 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1638** | 16kB | 1 | 32kB | 1 | 96 | 128 | 64 | 64 | 8 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1639** | 64kB | 4 | 4kB | 1 | 50 | 128 | 64 | 16 | 32 | 128 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1640** | 64kB | 1 | 64kB | 1 | 128 | 128 | 64 | 8 | 16 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1641** | 64kB | 1 | 16kB | 1 | 64 | 96 | 64 | 64 | 64 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1642** | 32kB | 2 | 16kB | 1 | 96 | 128 | 64 | 8 | 64 | 32 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1643** | 16kB | 4 | 4kB | 1 | 50 | 96 | 64 | 8 | 32 | 16 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1644** | 64kB | 1 | 8kB | 4 | 50 | 128 | 16 | 8 | 64 | 64 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1645** | 32kB | 1 | 32kB | 4 | 256 | 256 | 32 | 8 | 16 | 40 | 3 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1646** | 32kB | 2 | 16kB | 4 | 50 | 96 | 64 | 8 | 8 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1647** | 16kB | 4 | 16kB | 4 | 64 | 256 | 16 | 16 | 32 | 16 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1648** | 64kB | 4 | 16kB | 2 | 64 | 128 | 16 | 8 | 64 | 128 | 1 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1649** | 32kB | 2 | 8kB | 2 | 96 | 256 | 16 | 16 | 8 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1650** | 64kB | 4 | 8kB | 1 | 96 | 128 | 64 | 8 | 16 | 32 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1651** | 16kB | 2 | 8kB | 4 | 256 | 96 | 64 | 16 | 8 | 128 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1652** | 16kB | 4 | 8kB | 2 | 128 | 128 | 16 | 8 | 16 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1653** | 32kB | 1 | 32kB | 4 | 50 | 128 | 64 | 8 | 8 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1654** | 64kB | 1 | 16kB | 2 | 128 | 256 | 32 | 16 | 16 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1655** | 32kB | 2 | 32kB | 2 | 50 | 96 | 32 | 64 | 16 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1656** | 64kB | 1 | 32kB | 2 | 128 | 256 | 16 | 8 | 8 | 32 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1657** | 16kB | 4 | 4kB | 1 | 50 | 128 | 16 | 8 | 64 | 128 | 3 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1658** | 16kB | 1 | 32kB | 2 | 256 | 96 | 64 | 8 | 16 | 40 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1659** | 32kB | 2 | 8kB | 2 | 128 | 128 | 64 | 16 | 8 | 64 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1660** | 32kB | 4 | 16kB | 1 | 96 | 96 | 16 | 16 | 32 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1661** | 16kB | 1 | 4kB | 2 | 50 | 128 | 64 | 8 | 32 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1662** | 32kB | 2 | 64kB | 4 | 96 | 128 | 64 | 32 | 32 | 64 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1663** | 32kB | 2 | 64kB | 2 | 128 | 128 | 16 | 32 | 32 | 32 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1664** | 32kB | 1 | 16kB | 1 | 256 | 128 | 64 | 16 | 16 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1665** | 32kB | 2 | 64kB | 2 | 256 | 128 | 64 | 8 | 32 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1666** | 64kB | 1 | 32kB | 1 | 256 | 128 | 64 | 16 | 8 | 64 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1667** | 64kB | 2 | 4kB | 4 | 50 | 256 | 64 | 32 | 64 | 16 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1668** | 64kB | 1 | 16kB | 1 | 50 | 128 | 32 | 16 | 16 | 32 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1669** | 16kB | 2 | 64kB | 2 | 50 | 128 | 32 | 64 | 8 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1670** | 16kB | 2 | 16kB | 2 | 50 | 256 | 16 | 32 | 8 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1671** | 32kB | 4 | 64kB | 2 | 128 | 96 | 32 | 64 | 64 | 32 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1672** | 32kB | 1 | 4kB | 2 | 96 | 256 | 16 | 16 | 8 | 32 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1673** | 16kB | 4 | 8kB | 1 | 256 | 256 | 64 | 8 | 16 | 64 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1674** | 64kB | 2 | 16kB | 1 | 64 | 128 | 16 | 16 | 16 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1675** | 32kB | 2 | 16kB | 2 | 256 | 128 | 32 | 8 | 32 | 32 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1676** | 32kB | 2 | 8kB | 4 | 50 | 128 | 32 | 16 | 64 | 64 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1677** | 16kB | 4 | 32kB | 4 | 50 | 256 | 32 | 32 | 64 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1678** | 64kB | 2 | 4kB | 2 | 64 | 256 | 32 | 64 | 32 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1679** | 64kB | 1 | 64kB | 2 | 256 | 128 | 16 | 8 | 8 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1680** | 16kB | 2 | 64kB | 1 | 50 | 96 | 64 | 8 | 32 | 128 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1681** | 32kB | 1 | 4kB | 1 | 64 | 96 | 16 | 64 | 8 | 16 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | p. 134 |
| **#1682** | 64kB | 1 | 16kB | 2 | 96 | 128 | 64 | 64 | 8 | 40 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1683** | 16kB | 2 | 16kB | 2 | 50 | 96 | 16 | 32 | 8 | 40 | 1 | $2^{11}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1684** | 16kB | 2 | 4kB | 4 | 64 | 96 | 32 | 64 | 32 | 16 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1685** | 32kB | 2 | 64kB | 1 | 96 | 256 | 16 | 64 | 32 | 16 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1686** | 64kB | 2 | 16kB | 4 | 128 | 128 | 32 | 64 | 64 | 40 | 2 | $2^{11}$ | 1 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1687** | 16kB | 4 | 16kB | 4 | 50 | 128 | 32 | 16 | 32 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1688** | 16kB | 2 | 4kB | 4 | 128 | 128 | 64 | 16 | 8 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1689** | 32kB | 2 | 32kB | 2 | 96 | 96 | 16 | 8 | 64 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1690** | 16kB | 2 | 4kB | 2 | 256 | 256 | 16 | 32 | 16 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1691** | 16kB | 1 | 64kB | 1 | 256 | 128 | 16 | 16 | 32 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1692** | 16kB | 1 | 32kB | 2 | 128 | 96 | 64 | 64 | 8 | 40 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1693** | 64kB | 1 | 8kB | 1 | 96 | 128 | 16 | 16 | 64 | 16 | 2 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1694** | 64kB | 4 | 32kB | 1 | 50 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1695** | 64kB | 4 | 16kB | 4 | 64 | 128 | 64 | 16 | 8 | 32 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1696** | 16kB | 1 | 4kB | 4 | 50 | 128 | 16 | 64 | 32 | 32 | 1 | $2^{11}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1697** | 64kB | 2 | 4kB | 4 | 64 | 96 | 64 | 16 | 8 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1698** | 32kB | 2 | 16kB | 1 | 128 | 96 | 64 | 8 | 16 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1699** | 32kB | 4 | 4kB | 2 | 96 | 96 | 16 | 32 | 8 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1700** | 32kB | 1 | 8kB | 1 | 64 | 256 | 16 | 16 | 8 | 64 | 1 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1701** | 16kB | 4 | 4kB | 4 | 64 | 256 | 32 | 8 | 8 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1702** | 64kB | 2 | 16kB | 4 | 50 | 256 | 16 | 16 | 64 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1703** | 16kB | 2 | 4kB | 2 | 64 | 256 | 16 | 16 | 16 | 128 | 2 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1704** | 64kB | 1 | 64kB | 2 | 96 | 96 | 32 | 8 | 32 | 64 | 2 | $2^{14}$ | 1 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1705** | 64kB | 2 | 4kB | 1 | 64 | 256 | 32 | 64 | 16 | 32 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1706** | 16kB | 2 | 8kB | 2 | 128 | 128 | 64 | 8 | 64 | 40 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1707** | 16kB | 2 | 32kB | 4 | 50 | 256 | 16 | 64 | 8 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1708** | 64kB | 1 | 32kB | 1 | 64 | 128 | 64 | 8 | 16 | 64 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1709** | 64kB | 4 | 8kB | 4 | 256 | 96 | 64 | 16 | 64 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1710** | 16kB | 1 | 64kB | 1 | 96 | 96 | 32 | 64 | 8 | 128 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1711** | 32kB | 1 | 8kB | 4 | 96 | 256 | 64 | 64 | 32 | 40 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1712** | 32kB | 2 | 16kB | 4 | 256 | 96 | 16 | 32 | 64 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1713** | 64kB | 1 | 32kB | 1 | 64 | 96 | 16 | 32 | 32 | 40 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1714** | 64kB | 1 | 8kB | 1 | 96 | 128 | 32 | 8 | 16 | 16 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1715** | 16kB | 2 | 32kB | 4 | 64 | 256 | 32 | 32 | 64 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1716** | 64kB | 2 | 32kB | 1 | 128 | 96 | 64 | 8 | 64 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1717** | 16kB | 1 | 8kB | 2 | 50 | 96 | 16 | 16 | 64 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1718** | 16kB | 1 | 4kB | 4 | 256 | 128 | 16 | 32 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1719** | 32kB | 2 | 16kB | 2 | 256 | 96 | 64 | 64 | 16 | 128 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1720** | 16kB | 2 | 8kB | 1 | 256 | 256 | 16 | 8 | 16 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1721** | 16kB | 4 | 64kB | 4 | 64 | 256 | 16 | 32 | 8 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1722** | 64kB | 1 | 8kB | 1 | 96 | 128 | 16 | 32 | 32 | 40 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1723** | 16kB | 2 | 64kB | 2 | 128 | 96 | 32 | 8 | 64 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1724** | 32kB | 2 | 4kB | 1 | 256 | 96 | 16 | 32 | 32 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1725** | 64kB | 4 | 32kB | 2 | 64 | 256 | 16 | 32 | 64 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1726** | 64kB | 4 | 16kB | 4 | 256 | 96 | 64 | 16 | 8 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1727** | 32kB | 4 | 64kB | 2 | 128 | 128 | 32 | 8 | 32 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1728** | 64kB | 4 | 8kB | 4 | 64 | 96 | 32 | 8 | 8 | 16 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1729** | 16kB | 4 | 64kB | 2 | 128 | 256 | 32 | 8 | 32 | 16 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1730** | 32kB | 2 | 32kB | 1 | 64 | 256 | 32 | 8 | 16 | 16 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1731** | 32kB | 2 | 64kB | 2 | 96 | 128 | 32 | 32 | 16 | 64 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | p. 79 |
| **#1732** | 16kB | 4 | 8kB | 2 | 64 | 128 | 64 | 64 | 32 | 32 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1733** | 32kB | 4 | 32kB | 4 | 50 | 96 | 32 | 32 | 8 | 40 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#1734** | 16kB | 2 | 64kB | 2 | 256 | 96 | 64 | 64 | 16 | 64 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1735** | 32kB | 1 | 8kB | 4 | 256 | 128 | 32 | 32 | 16 | 16 | 3 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1736** | 64kB | 2 | 4kB | 2 | 50 | 256 | 16 | 16 | 64 | 16 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1737** | 16kB | 4 | 32kB | 4 | 50 | 96 | 16 | 8 | 16 | 40 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1738** | 16kB | 4 | 4kB | 1 | 256 | 128 | 16 | 8 | 16 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1739** | 32kB | 1 | 4kB | 2 | 128 | 96 | 32 | 64 | 16 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1740** | 64kB | 2 | 32kB | 2 | 64 | 96 | 64 | 64 | 8 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1741** | 64kB | 2 | 4kB | 2 | 50 | 128 | 16 | 16 | 16 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1742** | 64kB | 1 | 16kB | 1 | 256 | 96 | 32 | 64 | 16 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1743** | 16kB | 1 | 16kB | 2 | 64 | 96 | 64 | 16 | 8 | 40 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1744** | 32kB | 4 | 4kB | 4 | 128 | 96 | 16 | 32 | 16 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1745** | 16kB | 1 | 16kB | 2 | 64 | 256 | 32 | 32 | 64 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1746** | 32kB | 2 | 8kB | 4 | 96 | 96 | 16 | 64 | 16 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1747** | 32kB | 2 | 16kB | 1 | 64 | 128 | 64 | 64 | 32 | 64 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1748** | 16kB | 2 | 16kB | 4 | 128 | 256 | 64 | 32 | 64 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1749** | 64kB | 2 | 8kB | 1 | 50 | 96 | 16 | 64 | 8 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1750** | 16kB | 4 | 8kB | 2 | 96 | 256 | 64 | 64 | 8 | 32 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1751** | 64kB | 2 | 64kB | 4 | 64 | 128 | 16 | 8 | 16 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1752** | 64kB | 2 | 64kB | 1 | 128 | 128 | 64 | 32 | 8 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1753** | 64kB | 4 | 4kB | 4 | 256 | 256 | 32 | 8 | 64 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1754** | 32kB | 1 | 64kB | 2 | 256 | 256 | 64 | 8 | 64 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1755** | 16kB | 2 | 32kB | 2 | 256 | 256 | 32 | 32 | 16 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1756** | 32kB | 1 | 16kB | 1 | 50 | 256 | 32 | 32 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1757** | 16kB | 1 | 16kB | 1 | 50 | 128 | 32 | 8 | 32 | 40 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1758** | 64kB | 2 | 64kB | 2 | 128 | 96 | 32 | 16 | 16 | 40 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1759** | 32kB | 4 | 32kB | 4 | 50 | 256 | 16 | 8 | 64 | 32 | 2 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1760** | 64kB | 2 | 8kB | 1 | 64 | 128 | 32 | 8 | 16 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#1761** | 64kB | 1 | 64kB | 2 | 50 | 128 | 16 | 8 | 16 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1762** | 16kB | 4 | 8kB | 4 | 256 | 96 | 16 | 16 | 32 | 40 | 2 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1763** | 16kB | 4 | 16kB | 4 | 96 | 128 | 16 | 8 | 32 | 40 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1764** | 64kB | 1 | 32kB | 4 | 256 | 128 | 64 | 64 | 16 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1765** | 64kB | 4 | 32kB | 1 | 256 | 256 | 64 | 8 | 32 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1766** | 32kB | 2 | 64kB | 2 | 96 | 256 | 32 | 32 | 32 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | p. 143 |
| **#1767** | 16kB | 1 | 32kB | 4 | 50 | 128 | 32 | 64 | 32 | 40 | 1 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1768** | 16kB | 1 | 64kB | 4 | 50 | 128 | 32 | 8 | 8 | 40 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1769** | 16kB | 4 | 8kB | 4 | 96 | 128 | 16 | 8 | 16 | 40 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1770** | 32kB | 4 | 32kB | 2 | 64 | 128 | 64 | 8 | 16 | 128 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1771** | 64kB | 1 | 4kB | 2 | 128 | 256 | 32 | 16 | 8 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1772** | 16kB | 4 | 64kB | 2 | 64 | 256 | 32 | 32 | 16 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1773** | 16kB | 1 | 64kB | 4 | 256 | 128 | 32 | 8 | 8 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1774** | 16kB | 4 | 16kB | 4 | 96 | 256 | 32 | 8 | 8 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1775** | 64kB | 4 | 32kB | 2 | 256 | 128 | 64 | 8 | 16 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1776** | 64kB | 1 | 4kB | 2 | 64 | 128 | 64 | 32 | 32 | 128 | 1 | $2^{11}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1777** | 16kB | 2 | 4kB | 4 | 64 | 128 | 16 | 16 | 8 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1778** | 64kB | 1 | 32kB | 1 | 50 | 128 | 64 | 32 | 16 | 64 | 1 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1779** | 16kB | 4 | 16kB | 4 | 50 | 256 | 64 | 64 | 8 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1780** | 64kB | 1 | 16kB | 1 | 50 | 128 | 16 | 64 | 16 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1781** | 16kB | 2 | 32kB | 1 | 50 | 128 | 32 | 64 | 64 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1782** | 64kB | 1 | 8kB | 2 | 64 | 96 | 64 | 8 | 8 | 64 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1783** | 16kB | 1 | 16kB | 2 | 64 | 256 | 32 | 8 | 16 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1784** | 64kB | 2 | 64kB | 1 | 50 | 256 | 64 | 16 | 64 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1785** | 16kB | 2 | 16kB | 1 | 256 | 96 | 32 | 16 | 16 | 64 | 3 | $2^{14}$ | 1 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1786** | 32kB | 4 | 64kB | 4 | 64 | 128 | 16 | 8 | 32 | 32 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1787** | 16kB | 4 | 64kB | 4 | 50 | 128 | 64 | 8 | 16 | 32 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1788** | 64kB | 4 | 8kB | 2 | 64 | 128 | 16 | 64 | 64 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1789** | 32kB | 4 | 64kB | 4 | 64 | 128 | 32 | 32 | 8 | 32 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1790** | 16kB | 4 | 16kB | 4 | 96 | 96 | 32 | 64 | 64 | 128 | 2 | $2^{13}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1791** | 64kB | 4 | 8kB | 1 | 256 | 256 | 64 | 8 | 32 | 32 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1792** | 64kB | 2 | 64kB | 4 | 96 | 96 | 64 | 64 | 8 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1793** | 64kB | 4 | 8kB | 1 | 96 | 96 | 64 | 16 | 8 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1794** | 16kB | 2 | 32kB | 2 | 64 | 128 | 16 | 64 | 64 | 16 | 3 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1795** | 16kB | 2 | 16kB | 4 | 128 | 96 | 16 | 8 | 64 | 32 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1796** | 16kB | 4 | 4kB | 1 | 96 | 96 | 16 | 8 | 64 | 32 | 3 | $2^{10}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#1797** | 64kB | 1 | 4kB | 4 | 128 | 256 | 16 | 8 | 16 | 40 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1798** | 16kB | 1 | 16kB | 4 | 128 | 256 | 64 | 16 | 8 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1799** | 64kB | 2 | 8kB | 2 | 96 | 256 | 16 | 8 | 64 | 32 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1800** | 64kB | 2 | 8kB | 2 | 128 | 128 | 64 | 32 | 16 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1801** | 64kB | 4 | 4kB | 1 | 96 | 128 | 16 | 32 | 16 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1802** | 32kB | 2 | 64kB | 1 | 128 | 128 | 32 | 32 | 32 | 64 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1803** | 64kB | 1 | 64kB | 4 | 256 | 256 | 64 | 16 | 8 | 16 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1804** | 64kB | 2 | 32kB | 4 | 96 | 128 | 32 | 16 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1805** | 64kB | 1 | 8kB | 2 | 50 | 256 | 32 | 16 | 8 | 40 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1806** | 32kB | 2 | 4kB | 2 | 96 | 256 | 16 | 32 | 16 | 16 | 1 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1807** | 64kB | 1 | 8kB | 1 | 96 | 128 | 16 | 64 | 32 | 128 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1808** | 16kB | 1 | 32kB | 2 | 50 | 128 | 32 | 8 | 8 | 32 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1809** | 16kB | 1 | 8kB | 4 | 96 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1810** | 16kB | 2 | 4kB | 4 | 256 | 96 | 64 | 8 | 64 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1811** | 64kB | 4 | 8kB | 1 | 256 | 256 | 32 | 8 | 8 | 32 | 3 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1812** | 64kB | 4 | 8kB | 1 | 50 | 256 | 32 | 64 | 64 | 40 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1813** | 64kB | 1 | 64kB | 4 | 256 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1814** | 32kB | 4 | 8kB | 1 | 64 | 256 | 16 | 32 | 32 | 16 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1815** | 64kB | 1 | 8kB | 2 | 96 | 96 | 32 | 64 | 8 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1816** | 32kB | 4 | 64kB | 1 | 256 | 256 | 64 | 16 | 32 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#1817** | 64kB | 4 | 8kB | 1 | 96 | 128 | 64 | 8 | 64 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1818** | 32kB | 4 | 4kB | 1 | 64 | 256 | 16 | 8 | 64 | 16 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1819** | 32kB | 2 | 4kB | 1 | 96 | 96 | 16 | 16 | 8 | 128 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1820** | 16kB | 1 | 8kB | 1 | 96 | 96 | 32 | 32 | 32 | 40 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1821** | 64kB | 1 | 16kB | 2 | 64 | 128 | 32 | 64 | 32 | 40 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1822** | 64kB | 1 | 64kB | 4 | 128 | 96 | 32 | 64 | 16 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#1823** | 64kB | 4 | 64kB | 4 | 256 | 128 | 16 | 32 | 16 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1824** | 64kB | 4 | 16kB | 1 | 50 | 128 | 16 | 8 | 16 | 40 | 2 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1825** | 32kB | 4 | 16kB | 1 | 50 | 96 | 32 | 32 | 32 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1826** | 32kB | 4 | 64kB | 1 | 256 | 128 | 16 | 16 | 64 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1827** | 16kB | 2 | 16kB | 1 | 64 | 96 | 16 | 64 | 32 | 40 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1828** | 32kB | 2 | 8kB | 1 | 64 | 96 | 32 | 16 | 32 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1829** | 32kB | 2 | 32kB | 2 | 256 | 128 | 16 | 64 | 16 | 32 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1830** | 32kB | 1 | 4kB | 1 | 128 | 96 | 64 | 32 | 32 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1831** | 32kB | 2 | 16kB | 2 | 128 | 96 | 16 | 16 | 8 | 128 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1832** | 64kB | 1 | 8kB | 1 | 50 | 256 | 32 | 16 | 16 | 16 | 1 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1833** | 64kB | 1 | 16kB | 2 | 96 | 128 | 32 | 64 | 8 | 40 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1834** | 16kB | 1 | 64kB | 2 | 128 | 96 | 16 | 32 | 8 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1835** | 64kB | 2 | 4kB | 4 | 256 | 128 | 64 | 16 | 8 | 40 | 2 | $2^{12}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1836** | 32kB | 2 | 8kB | 4 | 64 | 96 | 64 | 64 | 32 | 64 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1837** | 32kB | 1 | 32kB | 2 | 96 | 96 | 64 | 8 | 16 | 40 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1838** | 16kB | 2 | 8kB | 4 | 64 | 256 | 16 | 16 | 8 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1839** | 32kB | 2 | 4kB | 2 | 64 | 128 | 16 | 32 | 64 | 128 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1840** | 64kB | 2 | 32kB | 1 | 64 | 128 | 64 | 8 | 16 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1841** | 16kB | 4 | 4kB | 4 | 96 | 96 | 32 | 64 | 32 | 16 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1842** | 16kB | 4 | 64kB | 1 | 50 | 96 | 64 | 8 | 8 | 32 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1843** | 64kB | 2 | 8kB | 1 | 64 | 128 | 32 | 16 | 8 | 40 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1844** | 64kB | 1 | 64kB | 1 | 64 | 256 | 64 | 64 | 32 | 128 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1845** | 32kB | 4 | 16kB | 1 | 128 | 256 | 32 | 16 | 8 | 64 | 2 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1846** | 16kB | 1 | 4kB | 2 | 64 | 96 | 32 | 32 | 16 | 40 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1847** | 64kB | 1 | 16kB | 1 | 96 | 128 | 64 | 8 | 16 | 64 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1848** | 32kB | 4 | 32kB | 1 | 50 | 128 | 16 | 32 | 32 | 128 | 1 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1849** | 64kB | 1 | 4kB | 4 | 64 | 96 | 16 | 16 | 64 | 40 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1850** | 16kB | 4 | 64kB | 2 | 96 | 96 | 64 | 16 | 16 | 32 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1851** | 16kB | 1 | 64kB | 4 | 128 | 128 | 64 | 16 | 64 | 16 | 2 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1852** | 32kB | 1 | 8kB | 4 | 256 | 96 | 32 | 64 | 8 | 40 | 1 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1853** | 32kB | 1 | 4kB | 1 | 128 | 96 | 32 | 64 | 8 | 40 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#1854** | 16kB | 4 | 32kB | 2 | 50 | 128 | 16 | 32 | 8 | 32 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1855** | 16kB | 4 | 64kB | 1 | 50 | 96 | 64 | 64 | 32 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1856** | 64kB | 1 | 16kB | 2 | 96 | 256 | 16 | 8 | 16 | 64 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1857** | 16kB | 4 | 32kB | 1 | 96 | 128 | 32 | 8 | 32 | 16 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1858** | 64kB | 1 | 8kB | 1 | 128 | 256 | 64 | 32 | 32 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1859** | 64kB | 1 | 32kB | 1 | 128 | 96 | 16 | 64 | 16 | 64 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1860** | 16kB | 2 | 32kB | 4 | 64 | 96 | 32 | 64 | 16 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1861** | 16kB | 2 | 8kB | 4 | 50 | 128 | 32 | 8 | 64 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1862** | 16kB | 2 | 4kB | 1 | 64 | 256 | 32 | 64 | 16 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1863** | 32kB | 4 | 16kB | 1 | 64 | 128 | 16 | 16 | 64 | 32 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1864** | 16kB | 4 | 4kB | 1 | 64 | 96 | 16 | 32 | 32 | 32 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1865** | 16kB | 2 | 8kB | 2 | 50 | 256 | 32 | 8 | 32 | 16 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1866** | 16kB | 2 | 8kB | 1 | 128 | 128 | 16 | 64 | 16 | 16 | 3 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1867** | 64kB | 4 | 16kB | 2 | 128 | 128 | 64 | 8 | 8 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1868** | 32kB | 1 | 4kB | 1 | 50 | 96 | 64 | 8 | 8 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1869** | 16kB | 2 | 64kB | 2 | 128 | 128 | 32 | 16 | 64 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1870** | 32kB | 2 | 64kB | 2 | 128 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#1871** | 16kB | 2 | 64kB | 1 | 96 | 256 | 16 | 16 | 8 | 64 | 1 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1872** | 64kB | 1 | 32kB | 2 | 50 | 128 | 64 | 64 | 64 | 32 | 1 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1873** | 32kB | 1 | 16kB | 4 | 64 | 256 | 64 | 64 | 16 | 64 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1874** | 64kB | 2 | 16kB | 2 | 64 | 128 | 32 | 64 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1875** | 32kB | 4 | 64kB | 4 | 64 | 128 | 16 | 8 | 16 | 128 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1876** | 32kB | 4 | 32kB | 2 | 256 | 128 | 64 | 8 | 64 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1877** | 16kB | 4 | 8kB | 4 | 50 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1878** | 32kB | 4 | 8kB | 1 | 96 | 96 | 64 | 8 | 64 | 16 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1879** | 16kB | 1 | 32kB | 1 | 50 | 96 | 64 | 8 | 8 | 64 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1880** | 16kB | 2 | 8kB | 4 | 256 | 256 | 32 | 8 | 32 | 64 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1881** | 64kB | 2 | 8kB | 2 | 50 | 128 | 64 | 64 | 32 | 64 | 1 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1882** | 16kB | 1 | 8kB | 1 | 128 | 256 | 16 | 16 | 16 | 16 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1883** | 64kB | 2 | 8kB | 2 | 50 | 96 | 32 | 32 | 64 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1884** | 64kB | 2 | 16kB | 4 | 50 | 96 | 16 | 16 | 16 | 32 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1885** | 64kB | 1 | 4kB | 2 | 128 | 128 | 16 | 8 | 8 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1886** | 64kB | 4 | 8kB | 4 | 256 | 128 | 16 | 16 | 64 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1887** | 32kB | 4 | 4kB | 1 | 96 | 128 | 16 | 64 | 64 | 64 | 3 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1888** | 16kB | 4 | 32kB | 2 | 128 | 128 | 64 | 32 | 32 | 40 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1889** | 16kB | 4 | 32kB | 4 | 256 | 128 | 32 | 16 | 8 | 16 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1890** | 32kB | 2 | 8kB | 2 | 50 | 256 | 16 | 64 | 8 | 128 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1891** | 32kB | 4 | 64kB | 1 | 256 | 128 | 16 | 32 | 16 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1892** | 64kB | 2 | 32kB | 4 | 96 | 128 | 32 | 64 | 64 | 32 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1893** | 64kB | 4 | 8kB | 1 | 64 | 128 | 32 | 16 | 8 | 32 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1894** | 32kB | 4 | 4kB | 4 | 96 | 128 | 16 | 8 | 32 | 64 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#1895** | 64kB | 2 | 16kB | 4 | 256 | 256 | 64 | 32 | 32 | 64 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1896** | 16kB | 2 | 4kB | 4 | 256 | 256 | 16 | 32 | 16 | 40 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1897** | 16kB | 1 | 64kB | 4 | 64 | 128 | 64 | 64 | 8 | 32 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1898** | 32kB | 4 | 16kB | 2 | 96 | 96 | 16 | 8 | 8 | 128 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1899** | 64kB | 4 | 32kB | 4 | 256 | 128 | 64 | 8 | 32 | 16 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#1900** | 16kB | 2 | 8kB | 2 | 50 | 96 | 32 | 64 | 8 | 64 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1901** | 32kB | 4 | 8kB | 1 | 256 | 96 | 16 | 16 | 8 | 40 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1902** | 32kB | 1 | 8kB | 1 | 64 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#1903** | 16kB | 4 | 64kB | 1 | 96 | 256 | 64 | 8 | 8 | 64 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#1904** | 32kB | 4 | 32kB | 4 | 96 | 96 | 32 | 8 | 32 | 16 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1905** | 16kB | 2 | 4kB | 4 | 128 | 128 | 64 | 16 | 16 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1906** | 16kB | 4 | 4kB | 2 | 64 | 96 | 16 | 64 | 16 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1907** | 16kB | 4 | 64kB | 1 | 128 | 96 | 32 | 16 | 64 | 32 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1908** | 32kB | 2 | 4kB | 4 | 50 | 96 | 64 | 16 | 32 | 128 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#1909** | 32kB | 1 | 4kB | 2 | 96 | 96 | 32 | 64 | 32 | 40 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1910** | 32kB | 2 | 64kB | 4 | 128 | 256 | 32 | 16 | 16 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#1911** | 32kB | 1 | 8kB | 2 | 50 | 256 | 16 | 8 | 16 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1912** | 64kB | 4 | 16kB | 1 | 50 | 128 | 64 | 32 | 64 | 64 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#1913** | 16kB | 1 | 32kB | 2 | 256 | 128 | 32 | 16 | 64 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1914** | 32kB | 1 | 64kB | 2 | 96 | 128 | 16 | 8 | 16 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1915** | 16kB | 4 | 32kB | 2 | 50 | 96 | 16 | 8 | 16 | 64 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#1916** | 64kB | 1 | 4kB | 1 | 128 | 96 | 32 | 16 | 8 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#1917** | 64kB | 2 | 8kB | 2 | 128 | 96 | 32 | 64 | 8 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#1918** | 64kB | 1 | 4kB | 2 | 50 | 256 | 16 | 32 | 32 | 16 | 3 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1919** | 16kB | 1 | 64kB | 2 | 64 | 256 | 16 | 64 | 64 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1920** | 32kB | 2 | 64kB | 2 | 50 | 256 | 64 | 16 | 16 | 32 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1921** | 16kB | 1 | 64kB | 4 | 50 | 96 | 16 | 16 | 8 | 32 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#1922** | 64kB | 2 | 64kB | 1 | 50 | 96 | 16 | 64 | 8 | 128 | 2 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#1923** | 64kB | 2 | 16kB | 2 | 64 | 128 | 16 | 8 | 64 | 128 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#1924** | 16kB | 2 | 4kB | 1 | 96 | 96 | 64 | 64 | 64 | 32 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1925** | 16kB | 1 | 4kB | 2 | 128 | 128 | 16 | 32 | 8 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1926** | 64kB | 4 | 64kB | 1 | 256 | 128 | 32 | 32 | 16 | 128 | 1 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1927** | 64kB | 2 | 16kB | 2 | 64 | 96 | 16 | 8 | 8 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#1928** | 32kB | 2 | 32kB | 4 | 96 | 96 | 64 | 64 | 16 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1929** | 16kB | 4 | 32kB | 1 | 50 | 96 | 16 | 64 | 8 | 16 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1930** | 64kB | 4 | 64kB | 4 | 50 | 256 | 64 | 32 | 8 | 128 | 2 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#1931** | 64kB | 4 | 4kB | 2 | 96 | 96 | 64 | 16 | 8 | 64 | 2 | $2^{13}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1932** | 64kB | 2 | 4kB | 1 | 50 | 256 | 16 | 64 | 8 | 128 | 1 | $2^{13}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1933** | 32kB | 1 | 4kB | 4 | 50 | 128 | 64 | 8 | 64 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#1934** | 32kB | 2 | 64kB | 2 | 50 | 128 | 64 | 8 | 16 | 128 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1935** | 16kB | 2 | 16kB | 1 | 64 | 256 | 64 | 32 | 64 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#1936** | 32kB | 4 | 4kB | 1 | 96 | 256 | 64 | 16 | 32 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#1937** | 16kB | 4 | 8kB | 2 | 128 | 96 | 64 | 64 | 32 | 16 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1938** | 16kB | 2 | 64kB | 1 | 128 | 128 | 16 | 64 | 64 | 40 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1939** | 16kB | 2 | 4kB | 4 | 256 | 128 | 32 | 64 | 32 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1940** | 16kB | 2 | 64kB | 4 | 128 | 256 | 16 | 16 | 16 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1941** | 32kB | 2 | 16kB | 4 | 64 | 96 | 64 | 64 | 16 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1942** | 64kB | 1 | 64kB | 4 | 50 | 128 | 64 | 64 | 32 | 128 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#1943** | 64kB | 1 | 4kB | 2 | 96 | 96 | 64 | 8 | 16 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1944** | 16kB | 2 | 4kB | 2 | 128 | 256 | 64 | 8 | 8 | 64 | 1 | $2^{11}$ | 3 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1945** | 64kB | 2 | 4kB | 1 | 50 | 128 | 16 | 8 | 64 | 32 | 2 | $2^{11}$ | 3 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#1946** | 32kB | 2 | 4kB | 1 | 50 | 96 | 64 | 64 | 64 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1947** | 64kB | 4 | 8kB | 1 | 96 | 128 | 32 | 8 | 16 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1948** | 32kB | 1 | 4kB | 1 | 256 | 256 | 32 | 64 | 32 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1949** | 64kB | 4 | 64kB | 4 | 128 | 256 | 32 | 16 | 64 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1950** | 16kB | 2 | 32kB | 1 | 128 | 256 | 16 | 64 | 32 | 64 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#1951** | 16kB | 4 | 16kB | 4 | 96 | 128 | 16 | 32 | 16 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#1952** | 16kB | 4 | 8kB | 4 | 50 | 128 | 16 | 64 | 64 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#1953** | 32kB | 1 | 16kB | 1 | 50 | 256 | 16 | 16 | 16 | 40 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#1954** | 32kB | 2 | 4kB | 2 | 128 | 256 | 16 | 16 | 16 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1955** | 32kB | 1 | 64kB | 1 | 128 | 96 | 16 | 32 | 64 | 32 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1956** | 32kB | 4 | 64kB | 2 | 96 | 96 | 64 | 32 | 32 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1957** | 16kB | 4 | 16kB | 1 | 50 | 96 | 64 | 16 | 8 | 32 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#1958** | 32kB | 2 | 16kB | 1 | 128 | 256 | 64 | 8 | 32 | 128 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1959** | 64kB | 4 | 16kB | 1 | 96 | 128 | 16 | 64 | 32 | 32 | 1 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1960** | 64kB | 2 | 8kB | 4 | 64 | 128 | 64 | 8 | 64 | 32 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1961** | 16kB | 1 | 16kB | 4 | 64 | 256 | 16 | 16 | 64 | 64 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1962** | 64kB | 2 | 64kB | 1 | 256 | 256 | 32 | 32 | 16 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#1963** | 32kB | 2 | 32kB | 2 | 64 | 96 | 64 | 32 | 8 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1964** | 32kB | 1 | 64kB | 2 | 256 | 96 | 32 | 16 | 32 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#1965** | 64kB | 4 | 4kB | 1 | 128 | 256 | 64 | 64 | 64 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1966** | 16kB | 1 | 16kB | 1 | 128 | 96 | 16 | 32 | 8 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#1967** | 32kB | 2 | 8kB | 1 | 128 | 96 | 32 | 8 | 16 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#1968** | 32kB | 2 | 16kB | 4 | 128 | 128 | 16 | 8 | 32 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#1969** | 64kB | 4 | 32kB | 1 | 64 | 256 | 64 | 8 | 64 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#1970** | 64kB | 2 | 32kB | 1 | 64 | 96 | 64 | 16 | 32 | 128 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1971** | 16kB | 4 | 16kB | 1 | 128 | 96 | 32 | 8 | 8 | 16 | 1 | $2^{14}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1972** | 64kB | 4 | 4kB | 1 | 128 | 128 | 64 | 64 | 16 | 16 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1973** | 16kB | 4 | 32kB | 2 | 96 | 96 | 64 | 32 | 16 | 64 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#1974** | 32kB | 4 | 8kB | 1 | 256 | 96 | 32 | 16 | 64 | 64 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1975** | 64kB | 1 | 64kB | 2 | 256 | 128 | 64 | 8 | 32 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#1976** | 64kB | 4 | 32kB | 1 | 50 | 96 | 16 | 32 | 16 | 128 | 2 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#1977** | 32kB | 1 | 32kB | 2 | 64 | 96 | 32 | 64 | 16 | 40 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#1978** | 32kB | 4 | 64kB | 2 | 64 | 256 | 32 | 16 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#1979** | 32kB | 1 | 4kB | 1 | 128 | 96 | 32 | 16 | 16 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#1980** | 32kB | 1 | 32kB | 2 | 50 | 256 | 32 | 16 | 8 | 40 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#1981** | 16kB | 2 | 64kB | 4 | 50 | 128 | 16 | 16 | 16 | 32 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | p. 142 |
| **#1982** | 64kB | 1 | 8kB | 4 | 128 | 96 | 32 | 64 | 32 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#1983** | 32kB | 4 | 64kB | 2 | 128 | 96 | 32 | 16 | 8 | 32 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#1984** | 64kB | 2 | 8kB | 2 | 256 | 96 | 64 | 64 | 64 | 128 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | p. 143 |
| **#1985** | 64kB | 2 | 16kB | 2 | 96 | 96 | 64 | 16 | 8 | 40 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1986** | 16kB | 4 | 8kB | 4 | 64 | 96 | 64 | 16 | 32 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#1987** | 16kB | 4 | 32kB | 1 | 64 | 96 | 16 | 64 | 16 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#1988** | 64kB | 1 | 64kB | 2 | 96 | 96 | 32 | 16 | 16 | 64 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#1989** | 32kB | 2 | 64kB | 4 | 256 | 128 | 64 | 8 | 8 | 128 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#1990** | 32kB | 4 | 64kB | 1 | 128 | 128 | 32 | 64 | 64 | 40 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#1991** | 16kB | 1 | 4kB | 2 | 256 | 96 | 64 | 64 | 8 | 128 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#1992** | 32kB | 1 | 32kB | 2 | 64 | 256 | 16 | 8 | 64 | 32 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#1993** | 32kB | 1 | 4kB | 1 | 256 | 96 | 64 | 8 | 32 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#1994** | 64kB | 2 | 64kB | 1 | 64 | 96 | 32 | 64 | 32 | 128 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#1995** | 64kB | 1 | 4kB | 4 | 64 | 128 | 32 | 64 | 64 | 40 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#1996** | 16kB | 4 | 4kB | 1 | 50 | 128 | 32 | 32 | 8 | 40 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#1997** | 16kB | 1 | 32kB | 2 | 128 | 96 | 16 | 64 | 64 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#1998** | 32kB | 1 | 64kB | 2 | 50 | 96 | 32 | 32 | 8 | 16 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#1999** | 16kB | 2 | 4kB | 1 | 64 | 128 | 32 | 8 | 32 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2000** | 16kB | 4 | 8kB | 2 | 50 | 128 | 16 | 32 | 32 | 32 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2001** | 32kB | 2 | 4kB | 4 | 96 | 128 | 32 | 64 | 16 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2002** | 64kB | 1 | 4kB | 1 | 128 | 96 | 16 | 16 | 8 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2003** | 16kB | 4 | 8kB | 4 | 128 | 256 | 16 | 32 | 64 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2004** | 32kB | 2 | 64kB | 4 | 64 | 256 | 64 | 32 | 64 | 32 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2005** | 32kB | 4 | 16kB | 4 | 256 | 96 | 32 | 32 | 16 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2006** | 16kB | 2 | 4kB | 4 | 50 | 96 | 32 | 32 | 16 | 64 | 2 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2007** | 64kB | 4 | 64kB | 1 | 50 | 96 | 32 | 8 | 64 | 32 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2008** | 32kB | 1 | 32kB | 2 | 256 | 128 | 32 | 8 | 32 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2009** | 16kB | 2 | 64kB | 2 | 128 | 96 | 32 | 64 | 8 | 64 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2010** | 32kB | 4 | 16kB | 4 | 64 | 96 | 32 | 8 | 64 | 128 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2011** | 32kB | 4 | 4kB | 2 | 256 | 96 | 64 | 32 | 32 | 32 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2012** | 64kB | 2 | 32kB | 1 | 96 | 128 | 32 | 16 | 8 | 16 | 2 | $2^{12}$ | 2 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2013** | 32kB | 1 | 32kB | 2 | 64 | 128 | 64 | 64 | 8 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2014** | 16kB | 1 | 64kB | 1 | 64 | 256 | 16 | 32 | 8 | 32 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2015** | 16kB | 1 | 64kB | 2 | 50 | 128 | 32 | 16 | 16 | 16 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2016** | 64kB | 2 | 64kB | 4 | 128 | 96 | 16 | 16 | 32 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2017** | 64kB | 2 | 16kB | 2 | 128 | 128 | 64 | 8 | 32 | 128 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2018** | 32kB | 2 | 4kB | 1 | 256 | 256 | 32 | 16 | 16 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2019** | 64kB | 1 | 16kB | 4 | 96 | 96 | 16 | 64 | 8 | 16 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2020** | 64kB | 1 | 64kB | 1 | 50 | 128 | 32 | 64 | 16 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2021** | 64kB | 2 | 64kB | 2 | 128 | 96 | 64 | 32 | 32 | 40 | 3 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2022** | 32kB | 1 | 64kB | 1 | 64 | 96 | 64 | 16 | 64 | 64 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2023** | 16kB | 2 | 64kB | 1 | 50 | 256 | 32 | 32 | 64 | 64 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2024** | 64kB | 4 | 64kB | 4 | 96 | 256 | 64 | 16 | 64 | 128 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2025** | 16kB | 1 | 4kB | 4 | 50 | 96 | 16 | 32 | 64 | 64 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2026** | 16kB | 2 | 64kB | 1 | 96 | 256 | 64 | 32 | 16 | 40 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2027** | 32kB | 2 | 4kB | 4 | 96 | 128 | 64 | 16 | 32 | 128 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | p. 79 |
| **#2028** | 32kB | 4 | 16kB | 4 | 128 | 128 | 16 | 64 | 32 | 128 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2029** | 32kB | 4 | 4kB | 1 | 96 | 256 | 64 | 16 | 32 | 128 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2030** | 64kB | 2 | 8kB | 4 | 64 | 128 | 32 | 64 | 8 | 128 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2031** | 32kB | 4 | 8kB | 2 | 50 | 96 | 64 | 32 | 32 | 64 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2032** | 64kB | 2 | 64kB | 4 | 64 | 256 | 32 | 64 | 32 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2033** | 32kB | 4 | 16kB | 2 | 96 | 128 | 16 | 16 | 8 | 40 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2034** | 16kB | 2 | 64kB | 4 | 50 | 256 | 32 | 32 | 64 | 40 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2035** | 64kB | 2 | 4kB | 2 | 256 | 96 | 16 | 8 | 16 | 128 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2036** | 64kB | 2 | 8kB | 1 | 128 | 128 | 32 | 16 | 32 | 40 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2037** | 32kB | 2 | 16kB | 1 | 50 | 256 | 64 | 16 | 64 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2038** | 32kB | 1 | 32kB | 1 | 128 | 128 | 16 | 32 | 64 | 64 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2039** | 64kB | 2 | 16kB | 4 | 96 | 96 | 16 | 64 | 8 | 64 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2040** | 16kB | 4 | 8kB | 1 | 96 | 256 | 64 | 8 | 64 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2041** | 32kB | 2 | 16kB | 4 | 64 | 128 | 32 | 32 | 64 | 32 | 3 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2042** | 64kB | 4 | 64kB | 2 | 96 | 96 | 64 | 32 | 16 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2043** | 32kB | 1 | 4kB | 1 | 96 | 96 | 16 | 8 | 32 | 40 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2044** | 32kB | 1 | 16kB | 2 | 64 | 256 | 32 | 8 | 32 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2045** | 16kB | 1 | 16kB | 2 | 256 | 96 | 64 | 8 | 32 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2046** | 32kB | 1 | 32kB | 4 | 64 | 128 | 64 | 32 | 8 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2047** | 32kB | 4 | 4kB | 4 | 128 | 128 | 64 | 64 | 32 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2[11]** | 16kB | 1 | 8kB | 2 | 50 | 256 | 16 | 64 | 64 | 40 | 3 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2049** | 16kB | 1 | 16kB | 1 | 128 | 96 | 16 | 64 | 64 | 32 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2050** | 16kB | 4 | 16kB | 4 | 64 | 128 | 32 | 32 | 16 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2051** | 16kB | 4 | 32kB | 1 | 64 | 128 | 32 | 8 | 8 | 128 | 2 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2052** | 32kB | 1 | 4kB | 1 | 64 | 128 | 32 | 8 | 64 | 128 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2053** | 64kB | 1 | 4kB | 1 | 64 | 96 | 16 | 64 | 32 | 16 | 3 | $2^{13}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2054** | 32kB | 1 | 16kB | 2 | 96 | 128 | 16 | 64 | 32 | 128 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2055** | 16kB | 1 | 32kB | 2 | 50 | 256 | 64 | 64 | 16 | 32 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2056** | 16kB | 2 | 16kB | 2 | 256 | 96 | 32 | 16 | 8 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2057** | 64kB | 1 | 64kB | 1 | 96 | 96 | 64 | 16 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2058** | 16kB | 2 | 64kB | 4 | 256 | 128 | 32 | 16 | 32 | 64 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2059** | 16kB | 1 | 32kB | 1 | 128 | 96 | 64 | 32 | 64 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2060** | 16kB | 1 | 4kB | 4 | 50 | 256 | 16 | 8 | 32 | 32 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2061** | 32kB | 1 | 8kB | 1 | 64 | 96 | 16 | 64 | 32 | 40 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2062** | 64kB | 4 | 8kB | 2 | 50 | 128 | 32 | 64 | 8 | 32 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2063** | 64kB | 1 | 64kB | 4 | 50 | 96 | 32 | 64 | 16 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2064** | 16kB | 4 | 32kB | 4 | 96 | 128 | 16 | 32 | 64 | 40 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2065** | 32kB | 1 | 64kB | 4 | 64 | 96 | 32 | 32 | 32 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2066** | 32kB | 2 | 32kB | 1 | 96 | 96 | 32 | 16 | 16 | 16 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2067** | 64kB | 2 | 64kB | 4 | 96 | 128 | 64 | 32 | 16 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2068** | 64kB | 4 | 64kB | 4 | 256 | 128 | 32 | 16 | 16 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2069** | 32kB | 4 | 64kB | 4 | 64 | 96 | 32 | 64 | 64 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2070** | 64kB | 4 | 32kB | 1 | 128 | 128 | 64 | 32 | 32 | 16 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2071** | 16kB | 1 | 64kB | 1 | 64 | 128 | 16 | 64 | 16 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2072** | 16kB | 2 | 32kB | 4 | 96 | 96 | 16 | 16 | 8 | 16 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2073** | 64kB | 2 | 64kB | 1 | 256 | 128 | 16 | 16 | 64 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2074** | 16kB | 2 | 64kB | 4 | 64 | 96 | 16 | 64 | 8 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2075** | 32kB | 4 | 32kB | 1 | 128 | 256 | 64 | 32 | 8 | 16 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2076** | 16kB | 1 | 4kB | 2 | 64 | 256 | 64 | 64 | 32 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2077** | 16kB | 4 | 4kB | 4 | 64 | 96 | 64 | 64 | 16 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2078** | 16kB | 1 | 32kB | 1 | 256 | 128 | 64 | 64 | 32 | 40 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2079** | 32kB | 4 | 8kB | 4 | 128 | 96 | 64 | 32 | 16 | 128 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2080** | 64kB | 2 | 64kB | 4 | 64 | 128 | 16 | 32 | 8 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2081** | 16kB | 1 | 4kB | 1 | 96 | 96 | 32 | 16 | 16 | 128 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2082** | 64kB | 4 | 8kB | 4 | 64 | 256 | 64 | 32 | 32 | 128 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2083** | 16kB | 1 | 16kB | 4 | 64 | 256 | 64 | 16 | 8 | 16 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2084** | 32kB | 2 | 32kB | 4 | 50 | 128 | 16 | 64 | 32 | 32 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2085** | 64kB | 1 | 4kB | 4 | 128 | 96 | 16 | 64 | 32 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2086** | 32kB | 2 | 4kB | 2 | 96 | 128 | 16 | 8 | 16 | 32 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2087** | 16kB | 1 | 32kB | 1 | 128 | 128 | 16 | 32 | 16 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2088** | 32kB | 1 | 4kB | 2 | 96 | 128 | 64 | 64 | 8 | 32 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2089** | 16kB | 4 | 4kB | 4 | 256 | 96 | 16 | 16 | 32 | 16 | 1 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2090** | 16kB | 2 | 8kB | 4 | 256 | 96 | 16 | 8 | 32 | 64 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2091** | 32kB | 4 | 4kB | 1 | 64 | 128 | 32 | 64 | 16 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2092** | 32kB | 1 | 4kB | 1 | 256 | 128 | 32 | 64 | 32 | 128 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2093** | 32kB | 4 | 16kB | 4 | 96 | 256 | 32 | 32 | 32 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2094** | 16kB | 1 | 16kB | 2 | 256 | 96 | 16 | 8 | 16 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2095** | 32kB | 1 | 32kB | 1 | 96 | 96 | 16 | 32 | 16 | 32 | 1 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2096** | 64kB | 2 | 16kB | 2 | 256 | 256 | 16 | 32 | 64 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2097** | 32kB | 1 | 64kB | 4 | 256 | 256 | 64 | 64 | 8 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2098** | 64kB | 2 | 16kB | 1 | 256 | 96 | 64 | 16 | 32 | 16 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2099** | 64kB | 2 | 32kB | 1 | 128 | 128 | 32 | 8 | 32 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2100** | 32kB | 1 | 16kB | 4 | 256 | 96 | 64 | 16 | 16 | 40 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2101** | 32kB | 4 | 4kB | 1 | 96 | 128 | 64 | 32 | 64 | 40 | 2 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2102** | 16kB | 4 | 32kB | 4 | 96 | 128 | 16 | 64 | 8 | 16 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2103** | 64kB | 2 | 16kB | 2 | 256 | 256 | 16 | 32 | 32 | 40 | 2 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2104** | 16kB | 2 | 4kB | 4 | 256 | 128 | 32 | 8 | 32 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2105** | 32kB | 2 | 4kB | 2 | 50 | 96 | 32 | 16 | 32 | 16 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2106** | 16kB | 2 | 16kB | 1 | 128 | 128 | 64 | 16 | 32 | 40 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2107** | 32kB | 4 | 16kB | 2 | 256 | 128 | 32 | 16 | 16 | 128 | 2 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2108** | 64kB | 4 | 8kB | 2 | 128 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2109** | 64kB | 1 | 4kB | 1 | 128 | 256 | 32 | 8 | 16 | 128 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2110** | 64kB | 1 | 4kB | 2 | 64 | 128 | 32 | 64 | 8 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2111** | 64kB | 2 | 8kB | 1 | 256 | 128 | 32 | 64 | 64 | 128 | 1 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2112** | 16kB | 1 | 32kB | 4 | 128 | 96 | 32 | 32 | 8 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2113** | 16kB | 1 | 32kB | 4 | 64 | 128 | 32 | 64 | 16 | 64 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2114** | 32kB | 1 | 32kB | 2 | 128 | 256 | 32 | 8 | 64 | 64 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2115** | 32kB | 1 | 8kB | 1 | 64 | 256 | 16 | 16 | 32 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2116** | 64kB | 2 | 16kB | 1 | 50 | 128 | 64 | 16 | 32 | 40 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2117** | 64kB | 1 | 32kB | 4 | 64 | 96 | 64 | 16 | 64 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2118** | 32kB | 1 | 32kB | 2 | 256 | 128 | 64 | 64 | 32 | 128 | 1 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2119** | 32kB | 1 | 32kB | 2 | 64 | 128 | 32 | 32 | 32 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2120** | 64kB | 1 | 64kB | 2 | 128 | 96 | 16 | 8 | 64 | 64 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2121** | 64kB | 1 | 64kB | 1 | 128 | 96 | 16 | 64 | 8 | 40 | 2 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2122** | 64kB | 2 | 8kB | 1 | 50 | 128 | 32 | 16 | 16 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2123** | 64kB | 2 | 16kB | 2 | 50 | 128 | 32 | 16 | 64 | 16 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2124** | 32kB | 4 | 64kB | 2 | 256 | 96 | 32 | 64 | 32 | 16 | 3 | $2^{13}$ | 1 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2125** | 64kB | 1 | 32kB | 1 | 256 | 96 | 64 | 32 | 16 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2126** | 64kB | 4 | 32kB | 2 | 96 | 96 | 32 | 32 | 8 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2127** | 64kB | 2 | 8kB | 4 | 50 | 96 | 64 | 8 | 8 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2128** | 64kB | 2 | 8kB | 1 | 50 | 256 | 32 | 8 | 64 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2129** | 32kB | 4 | 32kB | 4 | 128 | 256 | 32 | 64 | 16 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2130** | 16kB | 2 | 16kB | 4 | 96 | 256 | 16 | 16 | 32 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2131** | 64kB | 1 | 32kB | 2 | 50 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2132** | 32kB | 2 | 64kB | 2 | 50 | 96 | 32 | 8 | 32 | 16 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2133** | 64kB | 4 | 64kB | 2 | 64 | 256 | 16 | 64 | 16 | 40 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2134** | 32kB | 4 | 16kB | 1 | 128 | 256 | 16 | 32 | 32 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2135** | 64kB | 4 | 16kB | 4 | 256 | 256 | 16 | 8 | 8 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2136** | 64kB | 1 | 64kB | 1 | 128 | 96 | 32 | 32 | 8 | 40 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2137** | 32kB | 4 | 4kB | 1 | 96 | 256 | 16 | 8 | 16 | 16 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2138** | 64kB | 2 | 64kB | 4 | 96 | 128 | 16 | 8 | 16 | 32 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2139** | 32kB | 2 | 32kB | 4 | 50 | 96 | 64 | 16 | 8 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2140** | 32kB | 4 | 64kB | 1 | 96 | 256 | 32 | 32 | 8 | 32 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2141** | 32kB | 1 | 16kB | 4 | 50 | 128 | 64 | 64 | 32 | 32 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2142** | 64kB | 1 | 4kB | 1 | 64 | 96 | 16 | 16 | 16 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2143** | 16kB | 2 | 4kB | 1 | 128 | 128 | 64 | 8 | 32 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2144** | 32kB | 4 | 4kB | 2 | 96 | 256 | 16 | 8 | 8 | 32 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2145** | 32kB | 1 | 64kB | 2 | 256 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{13}$ | 3 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | p. 142 |
| **#2146** | 64kB | 1 | 32kB | 1 | 128 | 128 | 64 | 16 | 32 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2147** | 16kB | 4 | 64kB | 1 | 50 | 96 | 16 | 32 | 8 | 64 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2148** | 32kB | 4 | 32kB | 1 | 96 | 128 | 16 | 32 | 8 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2149** | 64kB | 2 | 4kB | 1 | 96 | 96 | 16 | 32 | 16 | 128 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2150** | 32kB | 1 | 4kB | 1 | 128 | 256 | 32 | 16 | 8 | 32 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2151** | 16kB | 4 | 32kB | 1 | 256 | 128 | 16 | 8 | 16 | 64 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2152** | 16kB | 4 | 8kB | 2 | 96 | 96 | 16 | 16 | 64 | 64 | 1 | $2^{12}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2153** | 32kB | 1 | 32kB | 4 | 96 | 96 | 16 | 32 | 32 | 16 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2154** | 64kB | 2 | 64kB | 1 | 256 | 128 | 16 | 64 | 64 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2155** | 16kB | 2 | 32kB | 2 | 96 | 128 | 64 | 8 | 8 | 32 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2156** | 16kB | 4 | 8kB | 2 | 50 | 128 | 16 | 8 | 32 | 32 | 2 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2157** | 64kB | 1 | 16kB | 1 | 50 | 128 | 32 | 64 | 64 | 128 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2158** | 16kB | 4 | 32kB | 2 | 64 | 96 | 64 | 64 | 16 | 16 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2159** | 16kB | 4 | 8kB | 2 | 64 | 96 | 64 | 8 | 8 | 32 | 3 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2160** | 64kB | 1 | 4kB | 4 | 256 | 256 | 32 | 32 | 32 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2161** | 32kB | 4 | 16kB | 2 | 128 | 128 | 16 | 8 | 64 | 128 | 2 | $2^{13}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2162** | 64kB | 2 | 4kB | 1 | 96 | 128 | 16 | 8 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2163** | 32kB | 1 | 16kB | 4 | 64 | 256 | 16 | 8 | 32 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2164** | 32kB | 4 | 8kB | 2 | 96 | 256 | 16 | 32 | 64 | 128 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2165** | 32kB | 1 | 64kB | 4 | 64 | 256 | 64 | 8 | 32 | 128 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2166** | 16kB | 4 | 8kB | 1 | 64 | 256 | 16 | 64 | 16 | 32 | 3 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2167** | 32kB | 2 | 64kB | 4 | 64 | 96 | 16 | 64 | 16 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2168** | 64kB | 4 | 4kB | 1 | 64 | 96 | 64 | 16 | 64 | 32 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2169** | 64kB | 2 | 8kB | 1 | 256 | 128 | 32 | 8 | 8 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2170** | 16kB | 2 | 4kB | 2 | 128 | 128 | 64 | 64 | 64 | 40 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2171** | 64kB | 1 | 4kB | 4 | 96 | 128 | 64 | 16 | 64 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2172** | 32kB | 1 | 32kB | 2 | 128 | 96 | 64 | 16 | 8 | 64 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2173** | 16kB | 4 | 16kB | 2 | 256 | 128 | 64 | 64 | 16 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2174** | 16kB | 1 | 32kB | 1 | 256 | 128 | 32 | 32 | 32 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2175** | 64kB | 2 | 8kB | 2 | 64 | 96 | 64 | 64 | 16 | 40 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2176** | 64kB | 4 | 32kB | 1 | 256 | 128 | 32 | 8 | 8 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2177** | 16kB | 1 | 16kB | 1 | 50 | 128 | 16 | 64 | 32 | 32 | 1 | $2^{11}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2178** | 64kB | 2 | 8kB | 4 | 50 | 128 | 64 | 8 | 32 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2179** | 16kB | 4 | 8kB | 4 | 64 | 96 | 64 | 8 | 8 | 40 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2180** | 64kB | 1 | 16kB | 1 | 50 | 96 | 32 | 8 | 32 | 40 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2181** | 32kB | 4 | 64kB | 2 | 256 | 128 | 64 | 16 | 16 | 128 | 2 | $2^{13}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2182** | 32kB | 2 | 16kB | 2 | 50 | 96 | 32 | 16 | 8 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2183** | 16kB | 4 | 8kB | 1 | 128 | 256 | 16 | 64 | 16 | 40 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2184** | 32kB | 2 | 8kB | 2 | 64 | 256 | 64 | 32 | 64 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2185** | 32kB | 4 | 64kB | 1 | 256 | 128 | 32 | 64 | 8 | 40 | 2 | $2^{13}$ | 3 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2186** | 64kB | 4 | 32kB | 2 | 128 | 128 | 32 | 32 | 32 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2187** | 32kB | 2 | 32kB | 1 | 128 | 128 | 32 | 32 | 8 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2188** | 32kB | 4 | 8kB | 1 | 128 | 128 | 64 | 8 | 32 | 16 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2189** | 16kB | 1 | 32kB | 1 | 128 | 256 | 32 | 32 | 32 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2190** | 64kB | 4 | 4kB | 1 | 128 | 96 | 32 | 8 | 16 | 32 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2191** | 16kB | 1 | 4kB | 1 | 128 | 128 | 16 | 32 | 32 | 32 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2192** | 32kB | 4 | 8kB | 1 | 256 | 128 | 32 | 64 | 64 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2193** | 16kB | 4 | 16kB | 1 | 128 | 128 | 32 | 64 | 16 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2194** | 16kB | 2 | 4kB | 2 | 256 | 96 | 16 | 64 | 64 | 32 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2195** | 32kB | 4 | 4kB | 4 | 128 | 96 | 64 | 8 | 64 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2196** | 64kB | 2 | 4kB | 1 | 128 | 256 | 64 | 8 | 16 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2197** | 64kB | 2 | 16kB | 4 | 64 | 128 | 32 | 64 | 16 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2198** | 16kB | 1 | 64kB | 2 | 50 | 256 | 16 | 32 | 8 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2199** | 32kB | 2 | 8kB | 1 | 256 | 96 | 64 | 64 | 8 | 16 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2200** | 16kB | 2 | 32kB | 1 | 128 | 256 | 64 | 16 | 8 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2201** | 16kB | 1 | 16kB | 1 | 64 | 128 | 32 | 64 | 32 | 40 | 1 | $2^{14}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2202** | 16kB | 1 | 16kB | 4 | 50 | 96 | 16 | 64 | 64 | 64 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2203** | 16kB | 1 | 4kB | 1 | 64 | 96 | 16 | 16 | 64 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2204** | 16kB | 4 | 4kB | 2 | 50 | 128 | 32 | 16 | 16 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2205** | 16kB | 4 | 32kB | 2 | 256 | 96 | 16 | 32 | 8 | 40 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2206** | 64kB | 2 | 8kB | 4 | 256 | 128 | 32 | 16 | 8 | 40 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2207** | 64kB | 2 | 16kB | 1 | 96 | 256 | 16 | 16 | 8 | 40 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2208** | 32kB | 1 | 64kB | 1 | 50 | 96 | 32 | 32 | 32 | 32 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2209** | 32kB | 4 | 64kB | 2 | 96 | 128 | 64 | 64 | 8 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2210** | 16kB | 2 | 64kB | 4 | 50 | 96 | 16 | 32 | 16 | 32 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2211** | 32kB | 1 | 16kB | 2 | 128 | 256 | 64 | 64 | 64 | 16 | 2 | $2^{13}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2212** | 32kB | 1 | 64kB | 4 | 96 | 128 | 16 | 16 | 8 | 64 | 3 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2213** | 16kB | 4 | 8kB | 2 | 128 | 96 | 16 | 64 | 32 | 128 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2214** | 16kB | 1 | 16kB | 4 | 50 | 96 | 64 | 64 | 64 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2215** | 16kB | 4 | 32kB | 1 | 256 | 256 | 32 | 32 | 8 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2216** | 64kB | 1 | 32kB | 1 | 96 | 128 | 16 | 64 | 32 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2217** | 16kB | 2 | 64kB | 2 | 50 | 128 | 32 | 16 | 16 | 64 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2218** | 32kB | 2 | 64kB | 1 | 256 | 128 | 32 | 8 | 32 | 32 | 3 | $2^{11}$ | 2 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2219** | 64kB | 2 | 32kB | 1 | 128 | 256 | 64 | 64 | 8 | 32 | 3 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2220** | 16kB | 4 | 64kB | 4 | 64 | 96 | 32 | 8 | 16 | 64 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2221** | 64kB | 4 | 16kB | 1 | 128 | 128 | 32 | 32 | 64 | 32 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2222** | 32kB | 1 | 8kB | 2 | 256 | 96 | 16 | 32 | 32 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2223** | 16kB | 2 | 4kB | 4 | 96 | 128 | 16 | 16 | 32 | 128 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2224** | 32kB | 1 | 64kB | 2 | 256 | 128 | 16 | 32 | 32 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2225** | 16kB | 1 | 16kB | 4 | 128 | 256 | 16 | 16 | 32 | 32 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2226** | 16kB | 1 | 32kB | 4 | 64 | 96 | 16 | 8 | 64 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2227** | 32kB | 2 | 8kB | 2 | 64 | 128 | 32 | 16 | 32 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2228** | 32kB | 2 | 4kB | 1 | 64 | 128 | 64 | 16 | 32 | 64 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2229** | 32kB | 4 | 4kB | 2 | 50 | 128 | 64 | 16 | 8 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2230** | 64kB | 2 | 8kB | 1 | 256 | 128 | 32 | 32 | 32 | 64 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2231** | 32kB | 1 | 16kB | 2 | 64 | 256 | 64 | 64 | 32 | 64 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2232** | 32kB | 2 | 8kB | 2 | 256 | 128 | 32 | 8 | 32 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2233** | 16kB | 1 | 32kB | 2 | 128 | 256 | 32 | 8 | 8 | 128 | 1 | $2^{11}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2234** | 16kB | 2 | 4kB | 4 | 96 | 256 | 16 | 8 | 16 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2235** | 32kB | 4 | 32kB | 4 | 128 | 96 | 64 | 8 | 64 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2236** | 64kB | 2 | 4kB | 2 | 256 | 96 | 16 | 8 | 32 | 16 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2237** | 32kB | 4 | 16kB | 2 | 96 | 256 | 32 | 64 | 64 | 40 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2238** | 16kB | 2 | 32kB | 2 | 256 | 96 | 16 | 8 | 16 | 128 | 3 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2239** | 64kB | 4 | 64kB | 1 | 256 | 256 | 64 | 16 | 64 | 16 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2240** | 64kB | 2 | 16kB | 1 | 128 | 96 | 64 | 64 | 32 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2241** | 16kB | 2 | 4kB | 4 | 50 | 128 | 64 | 16 | 8 | 128 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2242** | 32kB | 1 | 16kB | 2 | 50 | 128 | 32 | 8 | 16 | 64 | 2 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2243** | 16kB | 2 | 4kB | 4 | 64 | 256 | 64 | 16 | 32 | 64 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2244** | 16kB | 1 | 32kB | 1 | 50 | 256 | 16 | 8 | 16 | 40 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2245** | 32kB | 1 | 8kB | 1 | 96 | 96 | 64 | 32 | 8 | 64 | 3 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2246** | 64kB | 2 | 16kB | 4 | 256 | 128 | 64 | 8 | 8 | 16 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2247** | 16kB | 4 | 16kB | 4 | 128 | 96 | 32 | 32 | 16 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2248** | 64kB | 4 | 32kB | 4 | 128 | 96 | 16 | 32 | 8 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2249** | 32kB | 4 | 64kB | 2 | 50 | 96 | 32 | 64 | 32 | 32 | 2 | $2^{14}$ | 3 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2250** | 32kB | 2 | 8kB | 4 | 50 | 128 | 32 | 16 | 32 | 40 | 2 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2251** | 64kB | 4 | 16kB | 4 | 64 | 96 | 64 | 16 | 32 | 16 | 1 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2252** | 32kB | 2 | 4kB | 4 | 96 | 96 | 16 | 64 | 16 | 64 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2253** | 16kB | 2 | 32kB | 2 | 50 | 256 | 32 | 16 | 64 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2254** | 16kB | 2 | 4kB | 4 | 64 | 128 | 16 | 64 | 16 | 32 | 3 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2255** | 32kB | 2 | 8kB | 1 | 96 | 128 | 16 | 8 | 16 | 32 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2256** | 32kB | 1 | 4kB | 1 | 64 | 96 | 16 | 32 | 16 | 64 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2257** | 32kB | 1 | 8kB | 4 | 128 | 128 | 64 | 16 | 64 | 128 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2258** | 16kB | 2 | 4kB | 1 | 256 | 96 | 32 | 32 | 64 | 64 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2259** | 32kB | 1 | 8kB | 2 | 256 | 128 | 16 | 16 | 8 | 32 | 3 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2260** | 32kB | 4 | 8kB | 1 | 64 | 128 | 64 | 32 | 64 | 40 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2261** | 32kB | 2 | 4kB | 2 | 96 | 256 | 16 | 8 | 32 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2262** | 32kB | 2 | 4kB | 4 | 64 | 128 | 32 | 64 | 64 | 64 | 3 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2263** | 32kB | 1 | 64kB | 4 | 96 | 128 | 32 | 8 | 8 | 40 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2264** | 64kB | 4 | 64kB | 1 | 128 | 256 | 32 | 64 | 64 | 40 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2265** | 16kB | 4 | 8kB | 1 | 256 | 128 | 16 | 8 | 16 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2266** | 32kB | 4 | 4kB | 1 | 128 | 256 | 16 | 32 | 64 | 128 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2267** | 32kB | 1 | 32kB | 4 | 96 | 96 | 16 | 16 | 8 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2268** | 16kB | 1 | 8kB | 4 | 64 | 96 | 64 | 16 | 8 | 32 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2269** | 16kB | 1 | 8kB | 1 | 50 | 96 | 32 | 16 | 16 | 64 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2270** | 64kB | 1 | 32kB | 1 | 96 | 96 | 32 | 8 | 16 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2271** | 32kB | 1 | 16kB | 1 | 50 | 128 | 16 | 32 | 8 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2272** | 32kB | 2 | 8kB | 1 | 50 | 128 | 32 | 64 | 16 | 16 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2273** | 16kB | 1 | 16kB | 2 | 128 | 128 | 16 | 32 | 8 | 16 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2274** | 64kB | 4 | 32kB | 1 | 64 | 96 | 32 | 32 | 8 | 128 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2275** | 32kB | 2 | 4kB | 2 | 50 | 128 | 16 | 32 | 8 | 16 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2276** | 32kB | 2 | 8kB | 4 | 50 | 96 | 32 | 8 | 8 | 64 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2277** | 32kB | 2 | 4kB | 1 | 128 | 256 | 64 | 8 | 32 | 32 | 3 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2278** | 16kB | 1 | 64kB | 4 | 50 | 128 | 16 | 8 | 32 | 40 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2279** | 64kB | 1 | 4kB | 2 | 128 | 96 | 64 | 32 | 16 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2280** | 16kB | 1 | 64kB | 4 | 50 | 256 | 64 | 32 | 32 | 64 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2281** | 32kB | 1 | 4kB | 2 | 50 | 96 | 64 | 8 | 32 | 16 | 2 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2282** | 64kB | 1 | 64kB | 4 | 96 | 256 | 32 | 32 | 64 | 16 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2283** | 16kB | 1 | 64kB | 4 | 96 | 128 | 64 | 64 | 64 | 64 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2284** | 64kB | 2 | 64kB | 1 | 64 | 96 | 64 | 16 | 64 | 40 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2285** | 32kB | 4 | 8kB | 2 | 128 | 128 | 32 | 64 | 8 | 32 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2286** | 32kB | 1 | 64kB | 4 | 50 | 128 | 16 | 8 | 64 | 64 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2287** | 32kB | 4 | 32kB | 4 | 96 | 96 | 64 | 64 | 16 | 16 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2288** | 16kB | 1 | 64kB | 1 | 64 | 128 | 16 | 32 | 32 | 16 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | p. 142 |
| **#2289** | 16kB | 4 | 64kB | 2 | 256 | 96 | 64 | 16 | 16 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2290** | 64kB | 2 | 8kB | 4 | 256 | 256 | 16 | 8 | 64 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2291** | 16kB | 4 | 64kB | 4 | 64 | 96 | 64 | 32 | 8 | 32 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2292** | 32kB | 2 | 8kB | 1 | 96 | 128 | 32 | 16 | 32 | 64 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2293** | 32kB | 1 | 16kB | 4 | 128 | 96 | 64 | 64 | 8 | 40 | 1 | $2^{13}$ | 3 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2294** | 16kB | 2 | 8kB | 4 | 128 | 96 | 16 | 16 | 8 | 32 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2295** | 64kB | 2 | 8kB | 4 | 50 | 128 | 64 | 16 | 64 | 16 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2296** | 32kB | 4 | 8kB | 4 | 50 | 256 | 64 | 32 | 32 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2297** | 64kB | 1 | 64kB | 4 | 96 | 256 | 32 | 64 | 64 | 16 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2298** | 32kB | 1 | 64kB | 4 | 96 | 256 | 32 | 16 | 64 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2299** | 16kB | 2 | 32kB | 2 | 128 | 256 | 16 | 8 | 16 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2300** | 32kB | 4 | 32kB | 2 | 128 | 128 | 64 | 64 | 32 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2301** | 32kB | 2 | 32kB | 4 | 256 | 256 | 32 | 32 | 16 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2302** | 16kB | 2 | 32kB | 2 | 256 | 96 | 32 | 64 | 8 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2303** | 16kB | 4 | 16kB | 4 | 256 | 128 | 64 | 8 | 64 | 128 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2304** | 16kB | 4 | 16kB | 4 | 256 | 256 | 64 | 32 | 16 | 128 | 2 | $2^{13}$ | 1 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2305** | 64kB | 1 | 16kB | 4 | 128 | 128 | 64 | 64 | 64 | 40 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2306** | 32kB | 1 | 8kB | 4 | 50 | 256 | 16 | 64 | 32 | 128 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2307** | 32kB | 2 | 64kB | 1 | 64 | 256 | 16 | 8 | 64 | 40 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2308** | 32kB | 2 | 8kB | 4 | 128 | 96 | 16 | 16 | 64 | 64 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2309** | 32kB | 1 | 4kB | 1 | 96 | 128 | 16 | 8 | 16 | 32 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2310** | 64kB | 4 | 16kB | 2 | 128 | 256 | 16 | 16 | 16 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2311** | 64kB | 1 | 8kB | 2 | 50 | 128 | 16 | 16 | 8 | 32 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2312** | 32kB | 4 | 8kB | 4 | 50 | 256 | 64 | 32 | 32 | 64 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2313** | 16kB | 1 | 16kB | 2 | 96 | 96 | 32 | 8 | 8 | 32 | 1 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2314** | 64kB | 4 | 32kB | 4 | 256 | 96 | 16 | 64 | 8 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2315** | 16kB | 2 | 64kB | 1 | 64 | 256 | 32 | 16 | 64 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2316** | 32kB | 1 | 32kB | 4 | 128 | 128 | 32 | 8 | 64 | 64 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2317** | 16kB | 1 | 8kB | 2 | 96 | 256 | 32 | 32 | 16 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2318** | 16kB | 2 | 4kB | 4 | 50 | 96 | 64 | 16 | 64 | 128 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2319** | 16kB | 1 | 16kB | 1 | 96 | 96 | 32 | 32 | 32 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2320** | 32kB | 4 | 8kB | 2 | 128 | 256 | 32 | 8 | 16 | 64 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2321** | 32kB | 2 | 16kB | 1 | 64 | 256 | 64 | 8 | 64 | 64 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2322** | 32kB | 2 | 32kB | 4 | 96 | 96 | 16 | 8 | 64 | 64 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2323** | 16kB | 4 | 32kB | 1 | 50 | 128 | 64 | 8 | 8 | 32 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2324** | 16kB | 2 | 8kB | 1 | 96 | 96 | 64 | 64 | 32 | 128 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2325** | 16kB | 2 | 32kB | 4 | 50 | 96 | 32 | 16 | 8 | 32 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2326** | 32kB | 1 | 16kB | 2 | 50 | 128 | 32 | 64 | 16 | 32 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2327** | 64kB | 2 | 8kB | 1 | 256 | 96 | 16 | 16 | 32 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2328** | 64kB | 1 | 4kB | 1 | 96 | 128 | 64 | 32 | 64 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2329** | 16kB | 1 | 8kB | 4 | 256 | 96 | 32 | 32 | 16 | 128 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2330** | 32kB | 2 | 64kB | 1 | 64 | 256 | 16 | 8 | 16 | 128 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2331** | 32kB | 4 | 64kB | 1 | 128 | 128 | 32 | 16 | 16 | 128 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2332** | 32kB | 2 | 32kB | 2 | 64 | 256 | 32 | 16 | 64 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2333** | 32kB | 4 | 8kB | 2 | 256 | 256 | 16 | 64 | 32 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2334** | 16kB | 4 | 16kB | 4 | 96 | 128 | 16 | 8 | 32 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2335** | 16kB | 2 | 32kB | 1 | 96 | 96 | 32 | 32 | 16 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2336** | 64kB | 4 | 16kB | 1 | 96 | 96 | 64 | 8 | 16 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2337** | 32kB | 2 | 4kB | 2 | 256 | 128 | 16 | 64 | 64 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2338** | 32kB | 1 | 16kB | 1 | 256 | 96 | 32 | 32 | 16 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2339** | 32kB | 2 | 64kB | 1 | 64 | 96 | 16 | 64 | 8 | 16 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2340** | 32kB | 1 | 4kB | 4 | 96 | 96 | 32 | 16 | 16 | 16 | 3 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2341** | 32kB | 2 | 32kB | 2 | 256 | 128 | 64 | 16 | 64 | 32 | 1 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2342** | 64kB | 1 | 8kB | 4 | 128 | 96 | 16 | 8 | 8 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2343** | 16kB | 4 | 8kB | 4 | 128 | 256 | 16 | 32 | 8 | 64 | 1 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2344** | 64kB | 4 | 8kB | 4 | 128 | 256 | 16 | 16 | 16 | 16 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2345** | 16kB | 1 | 32kB | 1 | 96 | 256 | 16 | 8 | 32 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2346** | 64kB | 2 | 8kB | 4 | 64 | 128 | 16 | 16 | 32 | 40 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2347** | 32kB | 2 | 16kB | 4 | 96 | 128 | 16 | 32 | 32 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2348** | 16kB | 2 | 4kB | 1 | 96 | 128 | 16 | 64 | 64 | 64 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2349** | 16kB | 4 | 4kB | 2 | 50 | 96 | 32 | 32 | 8 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2350** | 64kB | 1 | 16kB | 4 | 64 | 96 | 64 | 64 | 64 | 16 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2351** | 64kB | 2 | 16kB | 1 | 256 | 128 | 16 | 8 | 8 | 40 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2352** | 32kB | 2 | 16kB | 2 | 96 | 96 | 16 | 64 | 16 | 128 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2353** | 64kB | 1 | 16kB | 4 | 256 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2354** | 16kB | 4 | 32kB | 2 | 50 | 128 | 64 | 8 | 16 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2355** | 32kB | 1 | 64kB | 4 | 50 | 96 | 16 | 32 | 8 | 128 | 3 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2356** | 16kB | 4 | 4kB | 2 | 64 | 128 | 64 | 64 | 32 | 16 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2357** | 32kB | 2 | 64kB | 2 | 128 | 128 | 16 | 16 | 64 | 128 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2358** | 32kB | 4 | 64kB | 2 | 64 | 128 | 64 | 64 | 64 | 32 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2359** | 32kB | 2 | 32kB | 1 | 50 | 96 | 16 | 16 | 32 | 40 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2360** | 32kB | 2 | 32kB | 2 | 96 | 128 | 16 | 64 | 16 | 128 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2361** | 32kB | 4 | 16kB | 1 | 256 | 128 | 32 | 32 | 32 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2362** | 16kB | 1 | 64kB | 1 | 128 | 96 | 64 | 32 | 8 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2363** | 16kB | 1 | 64kB | 4 | 64 | 256 | 64 | 8 | 64 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2364** | 16kB | 1 | 32kB | 1 | 64 | 96 | 32 | 8 | 8 | 16 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2365** | 32kB | 2 | 16kB | 2 | 256 | 256 | 64 | 32 | 32 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2366** | 16kB | 4 | 16kB | 4 | 128 | 96 | 64 | 16 | 16 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2367** | 16kB | 2 | 64kB | 1 | 96 | 128 | 16 | 64 | 8 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | p. 79 |
| **#2368** | 32kB | 1 | 16kB | 4 | 64 | 96 | 16 | 16 | 64 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2369** | 32kB | 4 | 4kB | 1 | 96 | 128 | 32 | 32 | 16 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2370** | 16kB | 2 | 64kB | 1 | 256 | 256 | 64 | 32 | 16 | 16 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2371** | 32kB | 1 | 4kB | 1 | 64 | 96 | 32 | 8 | 32 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2372** | 16kB | 1 | 8kB | 4 | 256 | 256 | 64 | 32 | 32 | 32 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2373** | 32kB | 2 | 64kB | 1 | 128 | 128 | 16 | 32 | 16 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2374** | 64kB | 1 | 8kB | 4 | 64 | 128 | 32 | 16 | 64 | 32 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2375** | 64kB | 1 | 4kB | 2 | 96 | 256 | 64 | 64 | 32 | 16 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2376** | 16kB | 4 | 16kB | 2 | 96 | 128 | 32 | 16 | 16 | 32 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2377** | 16kB | 1 | 64kB | 2 | 64 | 96 | 64 | 8 | 8 | 32 | 1 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2378** | 64kB | 1 | 4kB | 2 | 128 | 96 | 16 | 16 | 32 | 32 | 1 | $2^{11}$ | 2 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2379** | 64kB | 1 | 16kB | 2 | 256 | 128 | 64 | 8 | 64 | 16 | 2 | $2^{10}$ | 1 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2380** | 32kB | 1 | 32kB | 1 | 128 | 128 | 16 | 16 | 8 | 16 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2381** | 64kB | 2 | 32kB | 1 | 256 | 256 | 32 | 32 | 32 | 128 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2382** | 16kB | 1 | 32kB | 1 | 96 | 128 | 32 | 8 | 8 | 128 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2383** | 16kB | 1 | 8kB | 2 | 128 | 96 | 32 | 16 | 16 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2384** | 64kB | 1 | 8kB | 1 | 96 | 128 | 32 | 64 | 32 | 128 | 2 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2385** | 32kB | 4 | 16kB | 2 | 256 | 128 | 64 | 8 | 32 | 40 | 3 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2386** | 64kB | 2 | 8kB | 4 | 96 | 96 | 64 | 8 | 8 | 128 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2387** | 64kB | 2 | 64kB | 2 | 96 | 128 | 32 | 32 | 32 | 40 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2388** | 32kB | 4 | 64kB | 1 | 256 | 96 | 32 | 16 | 32 | 64 | 3 | $2^{11}$ | 2 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2389** | 32kB | 1 | 16kB | 2 | 64 | 128 | 64 | 64 | 8 | 32 | 3 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2390** | 32kB | 2 | 64kB | 1 | 256 | 128 | 64 | 32 | 16 | 64 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2391** | 64kB | 2 | 8kB | 2 | 256 | 128 | 16 | 8 | 32 | 40 | 1 | $2^{13}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2392** | 16kB | 1 | 8kB | 1 | 50 | 96 | 32 | 8 | 64 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2393** | 64kB | 4 | 8kB | 4 | 128 | 128 | 16 | 16 | 32 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2394** | 64kB | 2 | 64kB | 1 | 96 | 256 | 32 | 8 | 64 | 64 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2395** | 64kB | 1 | 16kB | 4 | 50 | 96 | 32 | 8 | 8 | 16 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2396** | 64kB | 1 | 16kB | 1 | 256 | 256 | 16 | 32 | 32 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2397** | 16kB | 2 | 16kB | 2 | 64 | 96 | 64 | 64 | 64 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2398** | 16kB | 1 | 4kB | 1 | 50 | 96 | 32 | 64 | 8 | 128 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2399** | 32kB | 2 | 4kB | 4 | 64 | 256 | 32 | 16 | 16 | 40 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2400** | 16kB | 2 | 16kB | 2 | 64 | 128 | 32 | 16 | 64 | 16 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2401** | 16kB | 2 | 8kB | 4 | 50 | 96 | 16 | 8 | 64 | 64 | 2 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2402** | 64kB | 1 | 16kB | 4 | 50 | 256 | 16 | 32 | 32 | 64 | 1 | $2^{14}$ | 1 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2403** | 16kB | 2 | 4kB | 4 | 96 | 128 | 32 | 64 | 64 | 128 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2404** | 16kB | 2 | 4kB | 2 | 256 | 96 | 16 | 8 | 32 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2405** | 32kB | 1 | 64kB | 1 | 50 | 256 | 64 | 64 | 64 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2406** | 16kB | 1 | 64kB | 1 | 64 | 256 | 64 | 8 | 32 | 64 | 2 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2407** | 32kB | 2 | 32kB | 2 | 256 | 128 | 64 | 16 | 8 | 32 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2408** | 32kB | 1 | 8kB | 2 | 96 | 256 | 64 | 64 | 8 | 40 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2409** | 64kB | 4 | 16kB | 4 | 96 | 128 | 16 | 8 | 8 | 32 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2410** | 64kB | 2 | 4kB | 2 | 50 | 256 | 32 | 8 | 64 | 128 | 2 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2411** | 32kB | 1 | 16kB | 4 | 256 | 96 | 64 | 32 | 64 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2412** | 32kB | 2 | 4kB | 2 | 256 | 96 | 32 | 64 | 16 | 64 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2413** | 32kB | 2 | 64kB | 2 | 256 | 96 | 16 | 64 | 32 | 32 | 3 | $2^{10}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2414** | 64kB | 1 | 4kB | 2 | 96 | 128 | 32 | 64 | 8 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2415** | 64kB | 2 | 4kB | 1 | 256 | 96 | 64 | 64 | 8 | 64 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2416** | 16kB | 2 | 16kB | 1 | 96 | 128 | 16 | 32 | 64 | 40 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | p. 81 |
| **#2417** | 16kB | 2 | 32kB | 1 | 64 | 128 | 16 | 16 | 32 | 64 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | p. 142 |
| **#2418** | 32kB | 1 | 32kB | 1 | 50 | 256 | 16 | 64 | 64 | 128 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2419** | 32kB | 1 | 8kB | 2 | 50 | 96 | 16 | 64 | 8 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2420** | 32kB | 2 | 64kB | 1 | 64 | 96 | 16 | 64 | 16 | 32 | 2 | $2^{13}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2421** | 64kB | 4 | 16kB | 2 | 64 | 96 | 16 | 8 | 16 | 128 | 2 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2422** | 32kB | 1 | 64kB | 2 | 64 | 96 | 64 | 8 | 64 | 128 | 3 | $2^{12}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2423** | 16kB | 1 | 8kB | 4 | 256 | 128 | 64 | 64 | 32 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2424** | 64kB | 4 | 64kB | 1 | 50 | 128 | 32 | 16 | 32 | 40 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2425** | 16kB | 4 | 32kB | 4 | 128 | 128 | 32 | 16 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2426** | 16kB | 2 | 64kB | 1 | 128 | 128 | 32 | 16 | 8 | 64 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2427** | 64kB | 1 | 8kB | 4 | 128 | 96 | 64 | 32 | 32 | 16 | 1 | $2^{10}$ | 3 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2428** | 32kB | 1 | 32kB | 1 | 128 | 256 | 64 | 16 | 32 | 16 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2429** | 16kB | 4 | 16kB | 4 | 128 | 128 | 64 | 16 | 16 | 16 | 1 | $2^{10}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2430** | 64kB | 1 | 8kB | 1 | 50 | 128 | 64 | 8 | 64 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2431** | 16kB | 2 | 32kB | 4 | 96 | 256 | 64 | 64 | 32 | 16 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2432** | 64kB | 4 | 16kB | 2 | 50 | 96 | 32 | 32 | 32 | 40 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2433** | 64kB | 1 | 16kB | 2 | 128 | 96 | 32 | 8 | 8 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2434** | 16kB | 1 | 64kB | 4 | 128 | 256 | 16 | 16 | 16 | 16 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2435** | 64kB | 1 | 64kB | 2 | 64 | 256 | 64 | 8 | 32 | 40 | 1 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2436** | 64kB | 2 | 8kB | 2 | 64 | 96 | 32 | 32 | 64 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2437** | 16kB | 2 | 4kB | 1 | 128 | 96 | 16 | 64 | 16 | 16 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2438** | 32kB | 4 | 4kB | 2 | 128 | 128 | 64 | 64 | 8 | 64 | 1 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2439** | 64kB | 1 | 64kB | 4 | 128 | 128 | 16 | 8 | 8 | 32 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2440** | 32kB | 4 | 4kB | 4 | 50 | 96 | 16 | 16 | 8 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2441** | 32kB | 2 | 32kB | 1 | 128 | 128 | 16 | 64 | 16 | 16 | 1 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2442** | 32kB | 4 | 8kB | 1 | 256 | 96 | 16 | 16 | 64 | 128 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2443** | 32kB | 1 | 16kB | 1 | 96 | 256 | 16 | 32 | 64 | 64 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2444** | 64kB | 4 | 32kB | 4 | 256 | 256 | 64 | 32 | 16 | 16 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2445** | 16kB | 2 | 16kB | 2 | 256 | 128 | 32 | 64 | 32 | 16 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2446** | 16kB | 2 | 4kB | 2 | 50 | 128 | 32 | 32 | 8 | 64 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2447** | 16kB | 2 | 4kB | 4 | 64 | 128 | 16 | 32 | 16 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2448** | 32kB | 4 | 16kB | 1 | 96 | 96 | 32 | 8 | 8 | 128 | 1 | $2^{10}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2449** | 32kB | 2 | 16kB | 2 | 128 | 128 | 32 | 64 | 8 | 64 | 3 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2450** | 16kB | 4 | 4kB | 2 | 64 | 128 | 16 | 8 | 16 | 64 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2451** | 32kB | 2 | 4kB | 2 | 64 | 128 | 64 | 8 | 32 | 40 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2452** | 16kB | 1 | 16kB | 1 | 128 | 96 | 64 | 8 | 8 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2453** | 16kB | 1 | 8kB | 2 | 256 | 96 | 32 | 8 | 16 | 128 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2454** | 16kB | 2 | 64kB | 1 | 256 | 96 | 64 | 32 | 32 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2455** | 32kB | 1 | 64kB | 2 | 50 | 128 | 16 | 32 | 64 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2456** | 64kB | 2 | 64kB | 1 | 50 | 96 | 32 | 16 | 64 | 128 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2457** | 64kB | 2 | 32kB | 2 | 256 | 256 | 16 | 16 | 32 | 64 | 3 | $2^{11}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2458** | 32kB | 4 | 64kB | 4 | 256 | 256 | 16 | 16 | 16 | 32 | 1 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2459** | 32kB | 1 | 4kB | 2 | 256 | 96 | 32 | 16 | 8 | 64 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2460** | 32kB | 2 | 8kB | 2 | 96 | 128 | 32 | 16 | 32 | 128 | 1 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2461** | 16kB | 2 | 32kB | 4 | 128 | 128 | 16 | 32 | 64 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2462** | 32kB | 1 | 8kB | 2 | 96 | 96 | 16 | 16 | 16 | 16 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2463** | 64kB | 2 | 4kB | 1 | 256 | 256 | 16 | 64 | 16 | 40 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2464** | 32kB | 2 | 64kB | 4 | 50 | 128 | 32 | 64 | 8 | 16 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2465** | 32kB | 2 | 4kB | 4 | 50 | 128 | 32 | 64 | 8 | 40 | 2 | $2^{13}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2466** | 64kB | 4 | 4kB | 1 | 64 | 96 | 32 | 64 | 8 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2467** | 16kB | 2 | 32kB | 2 | 64 | 96 | 64 | 64 | 16 | 32 | 1 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2468** | 64kB | 2 | 32kB | 2 | 50 | 128 | 64 | 8 | 16 | 128 | 2 | $2^{14}$ | 2 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2469** | 64kB | 1 | 16kB | 2 | 128 | 256 | 32 | 32 | 16 | 64 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2470** | 32kB | 1 | 64kB | 2 | 64 | 96 | 16 | 16 | 64 | 16 | 2 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2471** | 64kB | 2 | 16kB | 2 | 64 | 256 | 16 | 16 | 64 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2472** | 16kB | 4 | 8kB | 1 | 96 | 128 | 64 | 64 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2473** | 16kB | 4 | 4kB | 1 | 64 | 96 | 64 | 16 | 64 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2474** | 32kB | 1 | 4kB | 2 | 64 | 128 | 16 | 16 | 16 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2475** | 64kB | 4 | 4kB | 1 | 96 | 96 | 32 | 8 | 8 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2476** | 64kB | 4 | 32kB | 2 | 50 | 128 | 32 | 16 | 16 | 128 | 3 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2477** | 16kB | 4 | 64kB | 2 | 64 | 256 | 64 | 64 | 8 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2478** | 32kB | 4 | 8kB | 4 | 64 | 96 | 32 | 64 | 64 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2479** | 64kB | 4 | 8kB | 4 | 256 | 256 | 64 | 64 | 8 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2480** | 32kB | 2 | 64kB | 1 | 96 | 96 | 32 | 16 | 32 | 32 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2481** | 16kB | 2 | 4kB | 1 | 50 | 256 | 32 | 16 | 16 | 128 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2482** | 16kB | 4 | 32kB | 1 | 256 | 256 | 64 | 32 | 16 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2483** | 32kB | 1 | 16kB | 2 | 128 | 96 | 32 | 64 | 64 | 64 | 2 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2484** | 16kB | 1 | 64kB | 1 | 50 | 128 | 16 | 32 | 16 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2485** | 32kB | 1 | 4kB | 4 | 128 | 128 | 16 | 64 | 16 | 40 | 3 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2486** | 64kB | 1 | 64kB | 4 | 128 | 96 | 32 | 16 | 32 | 64 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2487** | 32kB | 1 | 64kB | 4 | 96 | 128 | 16 | 16 | 32 | 64 | 2 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2488** | 16kB | 4 | 32kB | 1 | 128 | 128 | 16 | 8 | 32 | 64 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2489** | 64kB | 4 | 16kB | 4 | 128 | 256 | 64 | 16 | 64 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2490** | 64kB | 2 | 64kB | 1 | 64 | 256 | 32 | 32 | 8 | 128 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2491** | 16kB | 4 | 32kB | 4 | 64 | 256 | 32 | 8 | 16 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2492** | 16kB | 1 | 32kB | 4 | 256 | 128 | 64 | 32 | 16 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2493** | 32kB | 1 | 8kB | 4 | 50 | 256 | 32 | 16 | 32 | 128 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2494** | 64kB | 2 | 64kB | 2 | 64 | 96 | 64 | 64 | 8 | 16 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2495** | 64kB | 1 | 64kB | 4 | 50 | 256 | 16 | 64 | 16 | 32 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2496** | 64kB | 1 | 8kB | 1 | 128 | 128 | 32 | 64 | 32 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2497** | 32kB | 4 | 32kB | 1 | 64 | 256 | 64 | 8 | 16 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2498** | 32kB | 4 | 32kB | 4 | 64 | 128 | 16 | 32 | 16 | 128 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2499** | 32kB | 2 | 32kB | 4 | 256 | 128 | 16 | 8 | 8 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2500** | 32kB | 1 | 4kB | 1 | 256 | 96 | 64 | 32 | 64 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2501** | 64kB | 4 | 16kB | 1 | 96 | 96 | 32 | 64 | 64 | 32 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2502** | 64kB | 4 | 32kB | 4 | 64 | 256 | 16 | 8 | 32 | 128 | 2 | $2^{11}$ | 1 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2503** | 64kB | 2 | 32kB | 2 | 128 | 128 | 32 | 32 | 32 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | p. 143 |
| **#2504** | 32kB | 1 | 32kB | 2 | 50 | 96 | 32 | 8 | 64 | 128 | 1 | $2^{12}$ | 2 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2505** | 32kB | 2 | 8kB | 4 | 64 | 96 | 32 | 8 | 8 | 128 | 3 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2506** | 64kB | 2 | 4kB | 1 | 50 | 96 | 64 | 64 | 64 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2507** | 64kB | 1 | 8kB | 1 | 50 | 128 | 64 | 64 | 8 | 32 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2508** | 64kB | 4 | 4kB | 1 | 96 | 96 | 32 | 32 | 32 | 16 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2509** | 64kB | 1 | 64kB | 4 | 256 | 128 | 16 | 64 | 8 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2510** | 16kB | 1 | 32kB | 2 | 256 | 256 | 32 | 64 | 16 | 64 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2511** | 16kB | 2 | 32kB | 1 | 50 | 128 | 32 | 64 | 32 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2512** | 16kB | 1 | 32kB | 2 | 64 | 96 | 64 | 8 | 8 | 32 | 1 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2513** | 32kB | 4 | 64kB | 4 | 96 | 96 | 32 | 8 | 16 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2514** | 64kB | 4 | 64kB | 4 | 96 | 256 | 64 | 8 | 8 | 40 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2515** | 64kB | 2 | 4kB | 2 | 256 | 256 | 32 | 32 | 32 | 64 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2516** | 32kB | 2 | 32kB | 4 | 256 | 96 | 32 | 64 | 8 | 128 | 2 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2517** | 32kB | 4 | 4kB | 4 | 256 | 256 | 16 | 16 | 16 | 128 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2518** | 32kB | 4 | 16kB | 4 | 96 | 128 | 32 | 32 | 32 | 64 | 1 | $2^{10}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2519** | 16kB | 4 | 8kB | 1 | 96 | 128 | 32 | 64 | 8 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2520** | 16kB | 4 | 64kB | 1 | 64 | 128 | 16 | 32 | 32 | 40 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2521** | 32kB | 4 | 4kB | 1 | 64 | 256 | 32 | 64 | 16 | 32 | 1 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2522** | 64kB | 4 | 32kB | 1 | 256 | 128 | 32 | 8 | 64 | 128 | 1 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2523** | 32kB | 2 | 16kB | 1 | 64 | 256 | 32 | 8 | 8 | 64 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2524** | 16kB | 2 | 8kB | 2 | 64 | 128 | 16 | 16 | 32 | 32 | 1 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | p. 79 |
| **#2525** | 16kB | 1 | 8kB | 2 | 96 | 128 | 32 | 16 | 16 | 32 | 1 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | p. 143 |
| **#2526** | 64kB | 2 | 64kB | 2 | 64 | 96 | 16 | 8 | 16 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2527** | 16kB | 4 | 32kB | 2 | 96 | 256 | 64 | 8 | 16 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2528** | 16kB | 4 | 16kB | 1 | 50 | 96 | 16 | 16 | 16 | 128 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2529** | 16kB | 2 | 64kB | 1 | 128 | 256 | 16 | 8 | 8 | 16 | 2 | $2^{13}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2530** | 64kB | 2 | 16kB | 4 | 96 | 256 | 32 | 16 | 64 | 64 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2531** | 32kB | 2 | 32kB | 2 | 256 | 256 | 16 | 8 | 8 | 128 | 2 | $2^{13}$ | 3 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2532** | 64kB | 1 | 16kB | 2 | 96 | 128 | 32 | 64 | 64 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2533** | 32kB | 2 | 16kB | 1 | 256 | 128 | 32 | 32 | 8 | 16 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2534** | 16kB | 2 | 64kB | 1 | 96 | 128 | 64 | 16 | 64 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2535** | 64kB | 1 | 32kB | 1 | 64 | 128 | 32 | 16 | 16 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2536** | 16kB | 1 | 8kB | 1 | 128 | 128 | 64 | 8 | 16 | 32 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2537** | 64kB | 2 | 64kB | 1 | 50 | 96 | 64 | 8 | 64 | 128 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2538** | 64kB | 4 | 64kB | 4 | 128 | 128 | 32 | 32 | 16 | 16 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2539** | 16kB | 2 | 4kB | 2 | 50 | 128 | 64 | 64 | 32 | 32 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 20 | |
| **#2540** | 16kB | 1 | 8kB | 1 | 256 | 256 | 64 | 16 | 64 | 64 | 1 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2541** | 16kB | 2 | 64kB | 4 | 256 | 256 | 64 | 64 | 64 | 128 | 1 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2542** | 16kB | 2 | 8kB | 1 | 50 | 256 | 64 | 32 | 32 | 16 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2543** | 16kB | 2 | 16kB | 1 | 96 | 256 | 16 | 8 | 64 | 40 | 3 | $2^{14}$ | 2 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2544** | 16kB | 4 | 8kB | 2 | 96 | 96 | 16 | 32 | 16 | 32 | 2 | $2^{13}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2545** | 16kB | 2 | 32kB | 1 | 96 | 256 | 64 | 16 | 64 | 64 | 2 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2546** | 64kB | 4 | 8kB | 1 | 64 | 128 | 16 | 64 | 32 | 64 | 2 | $2^{13}$ | 2 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2547** | 32kB | 4 | 32kB | 4 | 128 | 256 | 64 | 8 | 16 | 16 | 2 | $2^{10}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2548** | 16kB | 1 | 16kB | 2 | 96 | 256 | 64 | 64 | 16 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2549** | 32kB | 2 | 64kB | 2 | 50 | 96 | 32 | 32 | 8 | 40 | 3 | $2^{11}$ | 3 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2550** | 64kB | 1 | 64kB | 4 | 128 | 128 | 16 | 64 | 16 | 40 | 3 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2551** | 16kB | 1 | 4kB | 1 | 128 | 96 | 16 | 8 | 16 | 40 | 3 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2552** | 32kB | 1 | 32kB | 1 | 256 | 128 | 16 | 16 | 16 | 16 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2553** | 32kB | 4 | 64kB | 4 | 128 | 96 | 32 | 16 | 64 | 40 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2554** | 64kB | 1 | 64kB | 1 | 256 | 128 | 16 | 32 | 16 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2555** | 32kB | 4 | 8kB | 4 | 128 | 128 | 64 | 32 | 64 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2556** | 64kB | 2 | 8kB | 4 | 64 | 96 | 64 | 8 | 64 | 16 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2557** | 16kB | 4 | 32kB | 4 | 64 | 128 | 64 | 32 | 64 | 40 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2558** | 64kB | 2 | 8kB | 4 | 256 | 128 | 64 | 8 | 32 | 16 | 2 | $2^{10}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2559** | 16kB | 1 | 32kB | 1 | 50 | 256 | 16 | 32 | 16 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2560** | 64kB | 1 | 32kB | 2 | 64 | 128 | 64 | 64 | 32 | 64 | 2 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2561** | 32kB | 4 | 8kB | 1 | 128 | 96 | 16 | 16 | 64 | 40 | 3 | $2^{13}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2562** | 64kB | 1 | 64kB | 1 | 50 | 256 | 64 | 8 | 8 | 32 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2563** | 32kB | 2 | 4kB | 1 | 50 | 96 | 64 | 16 | 8 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2564** | 16kB | 1 | 32kB | 4 | 50 | 96 | 32 | 16 | 8 | 32 | 1 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2565** | 16kB | 4 | 64kB | 2 | 256 | 128 | 32 | 16 | 64 | 40 | 2 | $2^{10}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2566** | 16kB | 4 | 32kB | 4 | 128 | 128 | 64 | 64 | 64 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2567** | 64kB | 2 | 32kB | 4 | 256 | 256 | 16 | 16 | 64 | 32 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2568** | 64kB | 4 | 8kB | 1 | 96 | 96 | 64 | 32 | 16 | 128 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2569** | 64kB | 2 | 32kB | 4 | 50 | 256 | 16 | 64 | 8 | 40 | 3 | $2^{11}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2570** | 32kB | 4 | 16kB | 2 | 64 | 128 | 32 | 64 | 8 | 40 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2571** | 64kB | 2 | 16kB | 2 | 50 | 96 | 32 | 32 | 16 | 16 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2572** | 32kB | 1 | 64kB | 2 | 64 | 128 | 64 | 32 | 16 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2573** | 16kB | 1 | 64kB | 4 | 96 | 96 | 32 | 32 | 64 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2574** | 32kB | 2 | 8kB | 1 | 50 | 256 | 16 | 16 | 8 | 64 | 1 | $2^{12}$ | 2 | 10 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2575** | 32kB | 1 | 64kB | 2 | 64 | 128 | 32 | 16 | 64 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2576** | 64kB | 1 | 8kB | 2 | 96 | 128 | 32 | 32 | 16 | 64 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2577** | 64kB | 2 | 32kB | 4 | 96 | 128 | 64 | 16 | 32 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2578** | 16kB | 2 | 64kB | 1 | 50 | 128 | 16 | 32 | 8 | 64 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2579** | 64kB | 1 | 4kB | 2 | 96 | 256 | 32 | 8 | 16 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2580** | 64kB | 4 | 8kB | 2 | 96 | 96 | 16 | 8 | 32 | 32 | 1 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2581** | 64kB | 1 | 8kB | 4 | 96 | 96 | 32 | 8 | 8 | 40 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2582** | 64kB | 2 | 8kB | 4 | 96 | 128 | 16 | 8 | 8 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2583** | 32kB | 2 | 32kB | 1 | 256 | 128 | 16 | 64 | 8 | 64 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2584** | 32kB | 1 | 4kB | 2 | 50 | 256 | 32 | 64 | 32 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2585** | 16kB | 2 | 32kB | 1 | 96 | 128 | 64 | 64 | 16 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2586** | 64kB | 1 | 8kB | 2 | 64 | 96 | 32 | 16 | 64 | 128 | 3 | $2^{12}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2587** | 64kB | 2 | 32kB | 2 | 64 | 256 | 16 | 8 | 64 | 64 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2588** | 64kB | 2 | 8kB | 4 | 50 | 256 | 64 | 64 | 64 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2589** | 64kB | 1 | 64kB | 2 | 64 | 256 | 32 | 16 | 64 | 128 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2590** | 32kB | 4 | 16kB | 1 | 128 | 256 | 16 | 16 | 32 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2591** | 64kB | 4 | 32kB | 1 | 64 | 96 | 16 | 8 | 8 | 32 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2592** | 64kB | 1 | 32kB | 2 | 50 | 256 | 16 | 64 | 16 | 64 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2593** | 64kB | 2 | 32kB | 4 | 128 | 128 | 64 | 16 | 8 | 16 | 3 | $2^{10}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2594** | 64kB | 2 | 4kB | 4 | 256 | 256 | 32 | 32 | 64 | 128 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2595** | 64kB | 2 | 16kB | 4 | 64 | 256 | 32 | 16 | 8 | 16 | 2 | $2^{14}$ | 1 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2596** | 16kB | 4 | 4kB | 4 | 64 | 256 | 64 | 16 | 8 | 40 | 2 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2597** | 32kB | 2 | 8kB | 4 | 128 | 96 | 32 | 32 | 32 | 16 | 3 | $2^{12}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2598** | 32kB | 4 | 4kB | 2 | 128 | 96 | 32 | 16 | 16 | 32 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2599** | 64kB | 4 | 4kB | 1 | 96 | 256 | 32 | 8 | 8 | 128 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2600** | 16kB | 4 | 4kB | 2 | 50 | 96 | 16 | 32 | 64 | 128 | 3 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2601** | 32kB | 4 | 32kB | 4 | 256 | 256 | 16 | 64 | 32 | 16 | 2 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2602** | 32kB | 1 | 16kB | 1 | 50 | 96 | 64 | 16 | 16 | 128 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2603** | 64kB | 2 | 16kB | 1 | 256 | 128 | 32 | 16 | 64 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2604** | 64kB | 4 | 32kB | 2 | 256 | 256 | 16 | 32 | 16 | 40 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2605** | 16kB | 1 | 32kB | 1 | 256 | 96 | 32 | 16 | 32 | 32 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2606** | 16kB | 2 | 32kB | 4 | 128 | 128 | 64 | 64 | 64 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2607** | 64kB | 2 | 64kB | 1 | 96 | 96 | 16 | 32 | 16 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2608** | 64kB | 1 | 32kB | 4 | 96 | 256 | 64 | 64 | 16 | 128 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2609** | 64kB | 2 | 32kB | 2 | 128 | 256 | 16 | 16 | 32 | 16 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2610** | 64kB | 1 | 16kB | 4 | 256 | 256 | 16 | 32 | 64 | 32 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2611** | 32kB | 2 | 64kB | 2 | 256 | 96 | 32 | 32 | 16 | 40 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2612** | 16kB | 4 | 32kB | 2 | 96 | 96 | 64 | 16 | 8 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2613** | 64kB | 2 | 32kB | 4 | 50 | 96 | 32 | 64 | 32 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2614** | 16kB | 4 | 8kB | 4 | 96 | 96 | 32 | 32 | 32 | 128 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2615** | 32kB | 4 | 64kB | 2 | 256 | 128 | 32 | 64 | 16 | 16 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2616** | 32kB | 4 | 32kB | 4 | 50 | 96 | 64 | 64 | 32 | 16 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2617** | 16kB | 4 | 16kB | 1 | 50 | 96 | 16 | 64 | 64 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2618** | 32kB | 1 | 16kB | 4 | 128 | 128 | 32 | 16 | 8 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2619** | 32kB | 2 | 8kB | 4 | 96 | 96 | 64 | 16 | 64 | 64 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2620** | 32kB | 4 | 64kB | 4 | 256 | 256 | 16 | 64 | 64 | 128 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2621** | 32kB | 2 | 8kB | 1 | 256 | 256 | 16 | 16 | 64 | 40 | 3 | $2^{10}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2622** | 16kB | 2 | 16kB | 4 | 50 | 96 | 16 | 32 | 8 | 64 | 3 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2623** | 32kB | 4 | 8kB | 2 | 50 | 256 | 16 | 16 | 64 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2624** | 16kB | 4 | 8kB | 2 | 128 | 96 | 16 | 32 | 8 | 32 | 3 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2625** | 16kB | 4 | 4kB | 4 | 50 | 128 | 32 | 8 | 8 | 32 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2626** | 16kB | 4 | 4kB | 2 | 64 | 96 | 64 | 64 | 32 | 16 | 3 | $2^{11}$ | 1 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2627** | 64kB | 4 | 8kB | 1 | 96 | 256 | 64 | 16 | 32 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2628** | 64kB | 2 | 32kB | 1 | 256 | 128 | 64 | 8 | 32 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2629** | 32kB | 4 | 16kB | 1 | 96 | 96 | 16 | 16 | 16 | 32 | 3 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2630** | 16kB | 2 | 4kB | 1 | 64 | 128 | 32 | 8 | 32 | 16 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2631** | 32kB | 4 | 8kB | 4 | 50 | 256 | 16 | 64 | 32 | 40 | 3 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2632** | 64kB | 4 | 8kB | 4 | 96 | 256 | 16 | 64 | 64 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2633** | 64kB | 2 | 32kB | 4 | 50 | 256 | 32 | 32 | 32 | 40 | 1 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2634** | 16kB | 4 | 64kB | 1 | 50 | 96 | 16 | 8 | 16 | 40 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2635** | 64kB | 2 | 32kB | 1 | 64 | 96 | 16 | 32 | 64 | 40 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2636** | 64kB | 4 | 8kB | 4 | 128 | 256 | 16 | 32 | 64 | 32 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2637** | 32kB | 2 | 8kB | 1 | 96 | 96 | 32 | 32 | 16 | 16 | 1 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2638** | 64kB | 2 | 64kB | 2 | 50 | 256 | 64 | 32 | 8 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2639** | 64kB | 2 | 8kB | 2 | 128 | 128 | 64 | 64 | 32 | 16 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2640** | 32kB | 4 | 4kB | 1 | 64 | 128 | 32 | 16 | 8 | 64 | 3 | $2^{14}$ | 3 | 11 | $2^{11}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2641** | 64kB | 2 | 8kB | 1 | 256 | 128 | 16 | 32 | 8 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2642** | 64kB | 2 | 4kB | 1 | 64 | 96 | 16 | 16 | 64 | 40 | 2 | $2^{13}$ | 2 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2643** | 16kB | 4 | 32kB | 2 | 64 | 256 | 64 | 16 | 64 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2644** | 16kB | 4 | 16kB | 4 | 64 | 128 | 64 | 16 | 8 | 64 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2645** | 64kB | 1 | 8kB | 4 | 50 | 128 | 64 | 64 | 8 | 40 | 1 | $2^{13}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2646** | 32kB | 1 | 4kB | 2 | 128 | 128 | 32 | 8 | 64 | 128 | 3 | $2^{13}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2647** | 32kB | 1 | 64kB | 1 | 64 | 256 | 16 | 8 | 16 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2648** | 32kB | 1 | 16kB | 2 | 64 | 128 | 16 | 8 | 8 | 32 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2649** | 32kB | 4 | 64kB | 1 | 64 | 256 | 32 | 8 | 8 | 16 | 3 | $2^{12}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2650** | 32kB | 2 | 4kB | 2 | 64 | 128 | 16 | 8 | 8 | 40 | 2 | $2^{11}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2651** | 32kB | 2 | 32kB | 2 | 256 | 256 | 16 | 64 | 8 | 32 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2652** | 32kB | 1 | 32kB | 1 | 128 | 256 | 16 | 16 | 16 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2653** | 32kB | 2 | 16kB | 2 | 64 | 96 | 64 | 16 | 16 | 64 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2654** | 16kB | 2 | 64kB | 2 | 256 | 256 | 64 | 32 | 16 | 64 | 1 | $2^{10}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2655** | 16kB | 4 | 32kB | 1 | 64 | 128 | 32 | 16 | 16 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2656** | 16kB | 2 | 32kB | 2 | 256 | 128 | 16 | 32 | 8 | 40 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2657** | 64kB | 1 | 8kB | 2 | 50 | 256 | 64 | 64 | 16 | 32 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2658** | 32kB | 4 | 16kB | 4 | 96 | 128 | 16 | 8 | 64 | 64 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2659** | 32kB | 1 | 8kB | 4 | 96 | 128 | 32 | 32 | 64 | 40 | 3 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2660** | 16kB | 2 | 8kB | 1 | 256 | 96 | 16 | 32 | 64 | 128 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2661** | 16kB | 1 | 64kB | 1 | 64 | 128 | 64 | 8 | 16 | 64 | 2 | $2^{13}$ | 1 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2662** | 16kB | 1 | 32kB | 2 | 50 | 256 | 32 | 8 | 32 | 32 | 3 | $2^{11}$ | 3 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2663** | 64kB | 1 | 4kB | 2 | 64 | 256 | 32 | 16 | 32 | 40 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2664** | 64kB | 2 | 32kB | 4 | 128 | 128 | 32 | 32 | 8 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2665** | 64kB | 4 | 32kB | 4 | 128 | 128 | 64 | 8 | 8 | 64 | 2 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2666** | 64kB | 1 | 8kB | 2 | 64 | 256 | 64 | 64 | 32 | 32 | 1 | $2^{10}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2667** | 64kB | 2 | 8kB | 1 | 96 | 96 | 16 | 8 | 8 | 64 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2668** | 32kB | 2 | 4kB | 4 | 128 | 256 | 32 | 64 | 32 | 128 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2669** | 16kB | 4 | 32kB | 4 | 64 | 96 | 32 | 8 | 32 | 16 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2670** | 64kB | 1 | 64kB | 1 | 128 | 128 | 32 | 16 | 32 | 16 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2671** | 16kB | 4 | 8kB | 4 | 128 | 96 | 64 | 64 | 64 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2672** | 64kB | 1 | 64kB | 2 | 64 | 128 | 64 | 64 | 16 | 16 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2673** | 64kB | 2 | 64kB | 1 | 256 | 128 | 16 | 32 | 8 | 64 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2674** | 64kB | 2 | 64kB | 2 | 50 | 96 | 32 | 64 | 64 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2675** | 64kB | 1 | 16kB | 1 | 64 | 128 | 64 | 64 | 64 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2676** | 16kB | 4 | 4kB | 1 | 96 | 256 | 32 | 8 | 8 | 40 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2677** | 16kB | 2 | 4kB | 4 | 256 | 256 | 32 | 32 | 64 | 128 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2678** | 16kB | 4 | 64kB | 2 | 128 | 96 | 16 | 16 | 64 | 40 | 3 | $2^{14}$ | 1 | 11 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2679** | 16kB | 1 | 8kB | 2 | 256 | 96 | 32 | 8 | 32 | 16 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2680** | 16kB | 1 | 64kB | 1 | 64 | 256 | 64 | 32 | 16 | 16 | 1 | $2^{14}$ | 3 | 10 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2681** | 32kB | 1 | 32kB | 1 | 96 | 256 | 32 | 16 | 32 | 64 | 2 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2682** | 16kB | 2 | 16kB | 1 | 96 | 96 | 64 | 64 | 32 | 16 | 2 | $2^{12}$ | 3 | 10 | $2^{11}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2683** | 32kB | 4 | 64kB | 4 | 96 | 256 | 64 | 64 | 8 | 64 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2684** | 16kB | 4 | 16kB | 2 | 64 | 96 | 16 | 64 | 8 | 16 | 2 | $2^{14}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2685** | 64kB | 4 | 4kB | 4 | 64 | 96 | 64 | 64 | 16 | 40 | 1 | $2^{12}$ | 3 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2686** | 32kB | 4 | 32kB | 4 | 128 | 128 | 32 | 32 | 64 | 40 | 1 | $2^{10}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2687** | 32kB | 4 | 64kB | 1 | 50 | 256 | 64 | 16 | 64 | 32 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2688** | 16kB | 4 | 8kB | 4 | 128 | 128 | 16 | 16 | 16 | 40 | 1 | $2^{10}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2689** | 64kB | 2 | 8kB | 2 | 96 | 256 | 64 | 64 | 32 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2690** | 32kB | 1 | 8kB | 2 | 256 | 96 | 32 | 32 | 64 | 128 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2691** | 16kB | 1 | 64kB | 4 | 64 | 96 | 16 | 32 | 8 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2692** | 64kB | 4 | 4kB | 1 | 64 | 128 | 64 | 32 | 8 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2693** | 64kB | 1 | 64kB | 2 | 50 | 256 | 16 | 64 | 8 | 40 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2694** | 64kB | 2 | 16kB | 4 | 256 | 96 | 32 | 16 | 64 | 16 | 1 | $2^{12}$ | 3 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2695** | 64kB | 4 | 8kB | 1 | 256 | 96 | 16 | 8 | 8 | 16 | 3 | $2^{13}$ | 3 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2696** | 64kB | 1 | 32kB | 1 | 64 | 128 | 32 | 32 | 64 | 64 | 3 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2697** | 16kB | 4 | 16kB | 1 | 96 | 128 | 32 | 32 | 16 | 128 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2698** | 64kB | 1 | 16kB | 4 | 50 | 96 | 16 | 16 | 8 | 128 | 1 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2699** | 32kB | 1 | 4kB | 2 | 96 | 96 | 16 | 64 | 64 | 128 | 2 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2700** | 16kB | 2 | 16kB | 4 | 50 | 256 | 32 | 16 | 32 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2701** | 16kB | 1 | 4kB | 1 | 96 | 256 | 16 | 32 | 8 | 40 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2702** | 32kB | 1 | 32kB | 1 | 128 | 128 | 64 | 8 | 32 | 128 | 1 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2703** | 64kB | 4 | 32kB | 1 | 50 | 96 | 16 | 64 | 64 | 32 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2704** | 32kB | 1 | 8kB | 4 | 128 | 96 | 32 | 8 | 8 | 64 | 3 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2705** | 16kB | 1 | 32kB | 2 | 50 | 256 | 64 | 32 | 8 | 128 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2706** | 32kB | 1 | 8kB | 4 | 64 | 96 | 32 | 16 | 32 | 32 | 2 | $2^{14}$ | 3 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2707** | 32kB | 4 | 4kB | 4 | 50 | 96 | 32 | 32 | 8 | 128 | 2 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2708** | 64kB | 1 | 32kB | 4 | 256 | 96 | 32 | 16 | 8 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2709** | 64kB | 4 | 64kB | 1 | 256 | 256 | 64 | 8 | 16 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2710** | 16kB | 4 | 16kB | 1 | 256 | 96 | 64 | 8 | 32 | 40 | 2 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2711** | 32kB | 2 | 16kB | 4 | 50 | 128 | 64 | 16 | 32 | 64 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 1 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2712** | 32kB | 2 | 32kB | 2 | 64 | 96 | 16 | 16 | 8 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2713** | 64kB | 1 | 16kB | 4 | 128 | 96 | 32 | 16 | 8 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2714** | 16kB | 4 | 32kB | 4 | 64 | 128 | 32 | 64 | 8 | 32 | 1 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2715** | 64kB | 2 | 32kB | 1 | 128 | 96 | 32 | 64 | 64 | 32 | 3 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2716** | 16kB | 1 | 16kB | 2 | 50 | 256 | 32 | 8 | 8 | 16 | 1 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2717** | 16kB | 2 | 4kB | 2 | 50 | 128 | 16 | 16 | 32 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2718** | 32kB | 4 | 8kB | 1 | 50 | 128 | 32 | 8 | 16 | 128 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2719** | 32kB | 1 | 8kB | 4 | 256 | 256 | 32 | 32 | 32 | 32 | 2 | $2^{10}$ | 2 | 11 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2720** | 16kB | 4 | 16kB | 1 | 256 | 128 | 16 | 64 | 8 | 128 | 2 | $2^{10}$ | 2 | 11 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2721** | 16kB | 1 | 8kB | 1 | 128 | 256 | 64 | 16 | 64 | 16 | 2 | $2^{12}$ | 1 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2722** | 16kB | 4 | 8kB | 4 | 128 | 96 | 32 | 64 | 64 | 16 | 3 | $2^{12}$ | 3 | 10 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2723** | 16kB | 2 | 8kB | 2 | 256 | 96 | 64 | 64 | 16 | 128 | 2 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2724** | 16kB | 1 | 16kB | 4 | 96 | 128 | 32 | 16 | 64 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2725** | 32kB | 2 | 16kB | 1 | 96 | 128 | 16 | 64 | 16 | 128 | 3 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2726** | 64kB | 1 | 4kB | 4 | 256 | 128 | 16 | 64 | 16 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2727** | 64kB | 4 | 4kB | 4 | 50 | 256 | 32 | 16 | 16 | 64 | 2 | $2^{12}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2728** | 32kB | 1 | 64kB | 1 | 128 | 128 | 16 | 64 | 8 | 128 | 1 | $2^{14}$ | 3 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2729** | 64kB | 4 | 16kB | 4 | 64 | 96 | 64 | 16 | 32 | 40 | 1 | $2^{11}$ | 2 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2730** | 16kB | 1 | 8kB | 1 | 128 | 256 | 32 | 32 | 32 | 128 | 3 | $2^{10}$ | 1 | 10 | $2^{2}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2731** | 64kB | 1 | 64kB | 2 | 64 | 128 | 16 | 16 | 8 | 16 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2732** | 16kB | 1 | 16kB | 2 | 64 | 96 | 16 | 8 | 32 | 128 | 1 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2733** | 16kB | 1 | 32kB | 2 | 128 | 256 | 32 | 32 | 64 | 64 | 1 | $2^{10}$ | 1 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2734** | 64kB | 4 | 16kB | 2 | 50 | 96 | 32 | 64 | 16 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2735** | 64kB | 2 | 64kB | 1 | 50 | 96 | 16 | 16 | 16 | 128 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 18 | |
| **#2736** | 64kB | 1 | 64kB | 4 | 64 | 96 | 32 | 16 | 8 | 16 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2737** | 64kB | 4 | 16kB | 4 | 96 | 128 | 64 | 16 | 16 | 64 | 3 | $2^{14}$ | 3 | 11 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2738** | 32kB | 2 | 4kB | 4 | 50 | 256 | 16 | 64 | 32 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2739** | 64kB | 1 | 4kB | 1 | 128 | 256 | 64 | 8 | 32 | 40 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2740** | 32kB | 2 | 8kB | 4 | 256 | 128 | 32 | 64 | 32 | 128 | 2 | $2^{11}$ | 3 | 10 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2741** | 16kB | 1 | 8kB | 2 | 64 | 128 | 64 | 32 | 16 | 128 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2742** | 32kB | 4 | 4kB | 4 | 96 | 96 | 64 | 32 | 8 | 128 | 1 | $2^{11}$ | 3 | 11 | $2^{2}$ | 3 | $2^{11}$ | $2^{12}$ | 20 | |
| **#2743** | 32kB | 1 | 4kB | 2 | 50 | 256 | 16 | 32 | 32 | 40 | 3 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2744** | 64kB | 1 | 4kB | 1 | 128 | 256 | 64 | 32 | 32 | 128 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2745** | 16kB | 1 | 64kB | 2 | 50 | 128 | 32 | 8 | 32 | 64 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2746** | 64kB | 1 | 4kB | 4 | 256 | 96 | 32 | 8 | 16 | 16 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2747** | 64kB | 2 | 16kB | 4 | 64 | 256 | 32 | 16 | 32 | 64 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2748** | 64kB | 4 | 4kB | 1 | 256 | 96 | 64 | 8 | 8 | 128 | 2 | $2^{13}$ | 3 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2749** | 64kB | 1 | 4kB | 1 | 50 | 128 | 64 | 64 | 64 | 16 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2750** | 16kB | 4 | 4kB | 4 | 256 | 128 | 16 | 32 | 32 | 40 | 2 | $2^{12}$ | 2 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2751** | 16kB | 1 | 4kB | 2 | 64 | 256 | 64 | 32 | 64 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2752** | 32kB | 1 | 64kB | 4 | 96 | 256 | 16 | 8 | 16 | 128 | 1 | $2^{10}$ | 2 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2753** | 64kB | 4 | 16kB | 1 | 96 | 96 | 64 | 8 | 16 | 64 | 1 | $2^{11}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2754** | 64kB | 1 | 64kB | 1 | 128 | 128 | 32 | 32 | 32 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2755** | 16kB | 2 | 4kB | 1 | 256 | 256 | 32 | 64 | 64 | 16 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2756** | 64kB | 1 | 8kB | 4 | 256 | 96 | 32 | 32 | 32 | 16 | 3 | $2^{13}$ | 2 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2757** | 32kB | 1 | 16kB | 4 | 256 | 256 | 32 | 32 | 16 | 128 | 2 | $2^{11}$ | 3 | 10 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2758** | 16kB | 4 | 64kB | 1 | 64 | 96 | 16 | 32 | 8 | 128 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2759** | 16kB | 4 | 32kB | 1 | 256 | 128 | 32 | 32 | 32 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2760** | 16kB | 2 | 64kB | 2 | 64 | 96 | 64 | 16 | 8 | 64 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2761** | 16kB | 1 | 8kB | 1 | 96 | 128 | 16 | 16 | 16 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 18 | p. 142 |
| **#2762** | 64kB | 4 | 32kB | 1 | 128 | 96 | 16 | 16 | 16 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2763** | 16kB | 1 | 4kB | 4 | 128 | 256 | 64 | 8 | 32 | 40 | 1 | $2^{11}$ | 1 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2764** | 16kB | 4 | 64kB | 4 | 128 | 256 | 32 | 8 | 16 | 64 | 1 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2765** | 64kB | 4 | 8kB | 2 | 64 | 256 | 64 | 32 | 16 | 16 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2766** | 32kB | 2 | 64kB | 4 | 128 | 128 | 64 | 32 | 64 | 16 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2767** | 32kB | 4 | 32kB | 4 | 128 | 96 | 32 | 8 | 8 | 32 | 2 | $2^{10}$ | 1 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2768** | 64kB | 1 | 32kB | 2 | 96 | 96 | 32 | 64 | 32 | 32 | 3 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2769** | 32kB | 2 | 16kB | 4 | 256 | 96 | 64 | 64 | 8 | 16 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2770** | 16kB | 2 | 64kB | 4 | 256 | 128 | 16 | 64 | 16 | 128 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2771** | 64kB | 2 | 4kB | 1 | 96 | 128 | 32 | 8 | 8 | 32 | 2 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{10}$ | 20 | |
| **#2772** | 32kB | 1 | 32kB | 4 | 256 | 256 | 64 | 16 | 8 | 40 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2773** | 32kB | 1 | 8kB | 1 | 256 | 256 | 32 | 16 | 64 | 32 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2774** | 16kB | 1 | 64kB | 1 | 50 | 128 | 32 | 32 | 8 | 128 | 1 | $2^{14}$ | 2 | 11 | $2^{2}$ | 3 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2775** | 16kB | 2 | 64kB | 4 | 64 | 96 | 64 | 64 | 32 | 16 | 2 | $2^{12}$ | 1 | 10 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2776** | 16kB | 2 | 16kB | 1 | 96 | 128 | 32 | 8 | 32 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2777** | 16kB | 4 | 4kB | 2 | 256 | 96 | 32 | 64 | 64 | 16 | 2 | $2^{11}$ | 3 | 11 | $2^{10}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2778** | 16kB | 2 | 8kB | 4 | 64 | 96 | 16 | 64 | 8 | 16 | 3 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | p. 98 |
| **#2779** | 64kB | 2 | 32kB | 2 | 64 | 256 | 32 | 8 | 32 | 40 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2780** | 32kB | 4 | 32kB | 4 | 50 | 128 | 64 | 8 | 16 | 128 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2781** | 32kB | 4 | 4kB | 4 | 256 | 256 | 64 | 32 | 8 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2782** | 16kB | 4 | 32kB | 4 | 50 | 256 | 64 | 8 | 8 | 64 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2783** | 64kB | 2 | 64kB | 4 | 50 | 128 | 32 | 16 | 16 | 40 | 1 | $2^{11}$ | 1 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2784** | 64kB | 1 | 16kB | 2 | 50 | 128 | 16 | 8 | 64 | 64 | 1 | $2^{14}$ | 3 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2785** | 32kB | 1 | 4kB | 1 | 96 | 256 | 16 | 16 | 64 | 40 | 2 | $2^{13}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2786** | 32kB | 1 | 64kB | 4 | 128 | 256 | 16 | 64 | 32 | 128 | 1 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2787** | 16kB | 2 | 32kB | 2 | 96 | 96 | 32 | 8 | 64 | 40 | 1 | $2^{13}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2788** | 16kB | 1 | 4kB | 4 | 128 | 256 | 64 | 64 | 32 | 40 | 2 | $2^{14}$ | 2 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2789** | 32kB | 1 | 8kB | 1 | 256 | 128 | 64 | 8 | 32 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2790** | 64kB | 2 | 32kB | 1 | 128 | 256 | 32 | 32 | 8 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2791** | 64kB | 4 | 8kB | 2 | 128 | 96 | 16 | 16 | 64 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2792** | 16kB | 4 | 16kB | 1 | 96 | 96 | 64 | 64 | 64 | 128 | 1 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2793** | 16kB | 4 | 16kB | 1 | 128 | 128 | 64 | 8 | 64 | 32 | 1 | $2^{13}$ | 3 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2794** | 16kB | 4 | 64kB | 1 | 64 | 256 | 32 | 16 | 16 | 16 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2795** | 16kB | 1 | 4kB | 2 | 128 | 256 | 64 | 64 | 32 | 128 | 3 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2796** | 16kB | 2 | 64kB | 2 | 128 | 128 | 32 | 8 | 8 | 32 | 2 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2797** | 32kB | 2 | 32kB | 2 | 96 | 256 | 32 | 16 | 8 | 32 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2798** | 16kB | 1 | 32kB | 1 | 128 | 96 | 32 | 8 | 16 | 64 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2799** | 32kB | 4 | 8kB | 1 | 64 | 96 | 16 | 32 | 16 | 128 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2800** | 64kB | 2 | 32kB | 4 | 128 | 256 | 16 | 8 | 32 | 16 | 2 | $2^{11}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2801** | 16kB | 2 | 32kB | 4 | 50 | 96 | 16 | 8 | 8 | 16 | 2 | $2^{13}$ | 1 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 16 | |
| **#2802** | 64kB | 1 | 16kB | 4 | 50 | 256 | 32 | 32 | 16 | 128 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2803** | 16kB | 2 | 16kB | 4 | 64 | 256 | 16 | 16 | 32 | 32 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2804** | 16kB | 2 | 4kB | 2 | 64 | 96 | 16 | 32 | 32 | 16 | 2 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | p. 79 |
| **#2805** | 16kB | 2 | 16kB | 2 | 256 | 128 | 16 | 16 | 16 | 32 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2806** | 16kB | 4 | 32kB | 2 | 256 | 128 | 32 | 16 | 32 | 32 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2807** | 16kB | 1 | 16kB | 2 | 64 | 256 | 32 | 8 | 64 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2808** | 64kB | 1 | 32kB | 4 | 50 | 96 | 32 | 64 | 8 | 64 | 3 | $2^{11}$ | 2 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2809** | 16kB | 4 | 16kB | 2 | 128 | 96 | 64 | 8 | 64 | 40 | 3 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2810** | 64kB | 2 | 4kB | 4 | 64 | 128 | 64 | 64 | 32 | 64 | 1 | $2^{14}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2811** | 64kB | 1 | 64kB | 2 | 256 | 96 | 64 | 32 | 64 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2812** | 32kB | 4 | 4kB | 2 | 64 | 256 | 16 | 64 | 64 | 40 | 2 | $2^{12}$ | 3 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2813** | 64kB | 2 | 32kB | 1 | 50 | 256 | 32 | 64 | 8 | 40 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2814** | 16kB | 2 | 32kB | 1 | 64 | 256 | 16 | 8 | 8 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2815** | 32kB | 2 | 4kB | 4 | 256 | 128 | 64 | 64 | 32 | 40 | 2 | $2^{11}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2816** | 32kB | 1 | 16kB | 4 | 96 | 256 | 64 | 8 | 8 | 16 | 3 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2817** | 64kB | 2 | 32kB | 4 | 64 | 256 | 64 | 64 | 32 | 64 | 3 | $2^{12}$ | 2 | 12 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2818** | 32kB | 2 | 16kB | 1 | 64 | 128 | 64 | 16 | 64 | 32 | 2 | $2^{11}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2819** | 16kB | 2 | 16kB | 4 | 50 | 128 | 64 | 8 | 64 | 64 | 2 | $2^{12}$ | 2 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2820** | 64kB | 1 | 16kB | 4 | 128 | 128 | 64 | 8 | 8 | 128 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2821** | 64kB | 2 | 8kB | 2 | 64 | 256 | 32 | 64 | 64 | 64 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2822** | 64kB | 2 | 8kB | 1 | 128 | 96 | 64 | 64 | 64 | 16 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2823** | 32kB | 1 | 32kB | 4 | 256 | 256 | 16 | 32 | 8 | 16 | 2 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2824** | 64kB | 2 | 16kB | 4 | 128 | 96 | 64 | 64 | 32 | 40 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2825** | 32kB | 2 | 8kB | 1 | 96 | 256 | 32 | 64 | 16 | 32 | 3 | $2^{11}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2826** | 64kB | 2 | 8kB | 2 | 64 | 96 | 64 | 8 | 8 | 40 | 2 | $2^{14}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2827** | 16kB | 4 | 16kB | 2 | 64 | 96 | 64 | 8 | 64 | 64 | 2 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2828** | 32kB | 1 | 8kB | 2 | 64 | 256 | 32 | 32 | 16 | 64 | 1 | $2^{11}$ | 2 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2829** | 16kB | 2 | 16kB | 4 | 96 | 96 | 32 | 8 | 64 | 32 | 3 | $2^{14}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2830** | 64kB | 2 | 16kB | 4 | 50 | 256 | 32 | 8 | 16 | 16 | 2 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2831** | 16kB | 2 | 16kB | 1 | 256 | 128 | 16 | 8 | 8 | 32 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2832** | 64kB | 2 | 32kB | 2 | 256 | 256 | 64 | 32 | 64 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2833** | 32kB | 2 | 4kB | 2 | 256 | 256 | 64 | 64 | 8 | 16 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2834** | 64kB | 2 | 16kB | 1 | 128 | 96 | 16 | 32 | 32 | 40 | 3 | $2^{14}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 18 | |
| **#2835** | 64kB | 4 | 64kB | 4 | 50 | 128 | 16 | 32 | 16 | 64 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2836** | 32kB | 2 | 64kB | 2 | 50 | 128 | 32 | 64 | 64 | 64 | 2 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | p. 143 |
| **#2837** | 64kB | 1 | 64kB | 2 | 50 | 256 | 16 | 32 | 8 | 40 | 1 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2838** | 64kB | 4 | 32kB | 2 | 50 | 96 | 64 | 16 | 16 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2839** | 16kB | 1 | 64kB | 2 | 50 | 128 | 64 | 32 | 32 | 40 | 1 | $2^{14}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2840** | 32kB | 4 | 32kB | 4 | 50 | 128 | 32 | 16 | 16 | 40 | 1 | $2^{12}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2841** | 32kB | 1 | 4kB | 1 | 96 | 256 | 64 | 64 | 16 | 32 | 1 | $2^{13}$ | 2 | 12 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2842** | 16kB | 1 | 8kB | 4 | 64 | 96 | 64 | 32 | 64 | 16 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2843** | 64kB | 1 | 32kB | 2 | 64 | 128 | 32 | 16 | 32 | 32 | 1 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2844** | 32kB | 1 | 4kB | 2 | 64 | 128 | 64 | 32 | 8 | 40 | 1 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2845** | 16kB | 4 | 16kB | 1 | 96 | 96 | 32 | 64 | 16 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2846** | 16kB | 2 | 4kB | 2 | 128 | 128 | 64 | 64 | 64 | 16 | 2 | $2^{11}$ | 2 | 10 | $2^{10}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2847** | 32kB | 4 | 8kB | 1 | 128 | 96 | 16 | 8 | 64 | 128 | 3 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2848** | 16kB | 2 | 32kB | 1 | 96 | 96 | 64 | 32 | 8 | 16 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2849** | 32kB | 2 | 64kB | 2 | 64 | 96 | 32 | 64 | 32 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2850** | 16kB | 4 | 4kB | 1 | 128 | 96 | 16 | 32 | 32 | 64 | 2 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2851** | 64kB | 1 | 32kB | 2 | 256 | 256 | 32 | 8 | 32 | 16 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2852** | 16kB | 4 | 4kB | 2 | 50 | 128 | 16 | 8 | 8 | 32 | 1 | $2^{12}$ | 3 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2853** | 32kB | 4 | 16kB | 4 | 50 | 128 | 32 | 8 | 32 | 40 | 2 | $2^{10}$ | 2 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2854** | 16kB | 4 | 32kB | 1 | 96 | 256 | 64 | 16 | 32 | 128 | 2 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2855** | 64kB | 2 | 4kB | 2 | 96 | 128 | 64 | 32 | 8 | 40 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2856** | 32kB | 2 | 4kB | 4 | 64 | 96 | 64 | 64 | 64 | 32 | 2 | $2^{12}$ | 1 | 10 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 18 | |
| **#2857** | 32kB | 2 | 4kB | 2 | 50 | 128 | 16 | 8 | 8 | 128 | 2 | $2^{14}$ | 1 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 20 | |
| **#2858** | 16kB | 1 | 16kB | 2 | 64 | 256 | 16 | 64 | 32 | 128 | 1 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2859** | 64kB | 2 | 32kB | 4 | 256 | 256 | 32 | 64 | 16 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2860** | 64kB | 2 | 8kB | 2 | 50 | 128 | 64 | 8 | 32 | 16 | 3 | $2^{14}$ | 3 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2861** | 64kB | 2 | 16kB | 2 | 256 | 256 | 32 | 32 | 8 | 16 | 1 | $2^{12}$ | 3 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2862** | 32kB | 2 | 32kB | 4 | 128 | 256 | 16 | 64 | 16 | 16 | 3 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 20 | |
| **#2863** | 64kB | 4 | 64kB | 4 | 64 | 128 | 64 | 64 | 8 | 64 | 3 | $2^{12}$ | 3 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2864** | 32kB | 2 | 16kB | 4 | 50 | 256 | 32 | 8 | 32 | 40 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 1 | $2^{12}$ | $2^{11}$ | 18 | |
| **#2865** | 64kB | 4 | 64kB | 2 | 64 | 128 | 16 | 64 | 8 | 64 | 3 | $2^{12}$ | 2 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2866** | 64kB | 1 | 8kB | 2 | 128 | 128 | 64 | 8 | 64 | 16 | 2 | $2^{12}$ | 3 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2867** | 32kB | 1 | 16kB | 4 | 256 | 96 | 32 | 8 | 32 | 64 | 2 | $2^{11}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2868** | 32kB | 4 | 16kB | 2 | 50 | 128 | 64 | 64 | 16 | 128 | 2 | $2^{10}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{13}$ | 16 | |
| **#2869** | 16kB | 4 | 32kB | 1 | 96 | 128 | 64 | 8 | 64 | 16 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2870** | 16kB | 2 | 4kB | 1 | 50 | 256 | 32 | 8 | 8 | 40 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2871** | 32kB | 4 | 32kB | 4 | 64 | 256 | 32 | 32 | 8 | 16 | 3 | $2^{11}$ | 1 | 11 | $2^{11}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2872** | 64kB | 4 | 32kB | 1 | 64 | 96 | 64 | 16 | 16 | 32 | 1 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2873** | 16kB | 4 | 4kB | 4 | 256 | 256 | 32 | 32 | 16 | 128 | 2 | $2^{14}$ | 1 | 12 | $2^{11}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2874** | 16kB | 1 | 16kB | 1 | 256 | 128 | 64 | 32 | 32 | 64 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2875** | 64kB | 4 | 16kB | 4 | 96 | 128 | 64 | 16 | 32 | 64 | 3 | $2^{13}$ | 2 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2876** | 64kB | 1 | 64kB | 1 | 256 | 128 | 32 | 16 | 16 | 64 | 2 | $2^{12}$ | 2 | 10 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2877** | 16kB | 4 | 64kB | 2 | 64 | 96 | 16 | 8 | 64 | 64 | 2 | $2^{14}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2878** | 64kB | 1 | 32kB | 2 | 256 | 128 | 16 | 16 | 64 | 16 | 3 | $2^{10}$ | 3 | 10 | $2^{2}$ | 2 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2879** | 64kB | 4 | 32kB | 1 | 256 | 128 | 16 | 32 | 32 | 32 | 2 | $2^{12}$ | 3 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2880** | 32kB | 2 | 64kB | 4 | 96 | 96 | 32 | 16 | 16 | 32 | 1 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2881** | 64kB | 4 | 64kB | 2 | 50 | 128 | 16 | 8 | 32 | 32 | 2 | $2^{11}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2882** | 32kB | 4 | 8kB | 1 | 256 | 256 | 32 | 16 | 64 | 32 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2883** | 16kB | 1 | 16kB | 2 | 256 | 256 | 16 | 16 | 64 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2884** | 16kB | 4 | 64kB | 1 | 50 | 128 | 32 | 16 | 32 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{11}$ | 1 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2885** | 16kB | 4 | 16kB | 2 | 128 | 96 | 64 | 32 | 8 | 40 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{12}$ | 16 | |
| **#2886** | 32kB | 1 | 16kB | 2 | 64 | 128 | 16 | 32 | 16 | 16 | 3 | $2^{10}$ | 2 | 11 | $2^{11}$ | 1 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2887** | 16kB | 1 | 64kB | 2 | 256 | 128 | 64 | 16 | 8 | 64 | 3 | $2^{11}$ | 3 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2888** | 32kB | 1 | 8kB | 1 | 64 | 96 | 64 | 8 | 64 | 64 | 2 | $2^{14}$ | 3 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2889** | 64kB | 2 | 16kB | 4 | 256 | 96 | 32 | 32 | 16 | 40 | 2 | $2^{10}$ | 3 | 10 | $2^{11}$ | 3 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2890** | 32kB | 4 | 64kB | 2 | 64 | 96 | 16 | 64 | 64 | 32 | 3 | $2^{12}$ | 1 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2891** | 64kB | 2 | 16kB | 4 | 96 | 96 | 32 | 32 | 16 | 64 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2892** | 16kB | 1 | 64kB | 4 | 96 | 128 | 16 | 8 | 8 | 128 | 2 | $2^{10}$ | 3 | 12 | $2^{11}$ | 2 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2893** | 16kB | 4 | 4kB | 4 | 96 | 128 | 16 | 64 | 32 | 128 | 3 | $2^{14}$ | 2 | 10 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2894** | 16kB | 2 | 32kB | 4 | 128 | 128 | 16 | 32 | 64 | 32 | 1 | $2^{13}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2895** | 16kB | 4 | 16kB | 2 | 50 | 128 | 16 | 8 | 8 | 128 | 2 | $2^{11}$ | 1 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{10}$ | 18 | |
| **#2896** | 64kB | 4 | 32kB | 2 | 50 | 128 | 64 | 64 | 32 | 64 | 2 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2897** | 64kB | 1 | 16kB | 1 | 128 | 96 | 32 | 16 | 64 | 16 | 2 | $2^{10}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2898** | 32kB | 1 | 4kB | 2 | 64 | 256 | 32 | 8 | 16 | 32 | 1 | $2^{14}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |
| **#2899** | 32kB | 2 | 4kB | 4 | 96 | 256 | 16 | 64 | 8 | 128 | 2 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2900** | 32kB | 4 | 16kB | 4 | 96 | 128 | 16 | 16 | 16 | 64 | 3 | $2^{10}$ | 3 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2901** | 32kB | 2 | 4kB | 2 | 256 | 128 | 64 | 16 | 32 | 128 | 3 | $2^{10}$ | 3 | 11 | $2^{10}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2902** | 16kB | 2 | 64kB | 2 | 50 | 256 | 64 | 8 | 8 | 128 | 1 | $2^{10}$ | 1 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2903** | 16kB | 2 | 16kB | 1 | 128 | 128 | 16 | 64 | 64 | 128 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 18 | |
| **#2904** | 64kB | 2 | 64kB | 4 | 50 | 256 | 16 | 8 | 64 | 40 | 1 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{10}$ | 16 | |
| **#2905** | 16kB | 2 | 64kB | 2 | 50 | 128 | 16 | 32 | 16 | 40 | 2 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2906** | 16kB | 4 | 4kB | 1 | 256 | 96 | 64 | 16 | 64 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2907** | 16kB | 2 | 8kB | 2 | 256 | 256 | 16 | 8 | 16 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2908** | 64kB | 4 | 32kB | 1 | 96 | 96 | 32 | 32 | 64 | 32 | 3 | $2^{12}$ | 3 | 11 | $2^{2}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2909** | 16kB | 4 | 32kB | 2 | 50 | 128 | 32 | 16 | 8 | 64 | 1 | $2^{14}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2910** | 64kB | 1 | 64kB | 1 | 64 | 256 | 64 | 8 | 16 | 64 | 3 | $2^{13}$ | 1 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 20 | |
| **#2911** | 32kB | 4 | 32kB | 1 | 256 | 96 | 32 | 64 | 8 | 32 | 2 | $2^{10}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2912** | 32kB | 2 | 16kB | 2 | 256 | 128 | 16 | 16 | 32 | 32 | 1 | $2^{11}$ | 2 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2913** | 32kB | 4 | 64kB | 2 | 50 | 256 | 16 | 32 | 32 | 64 | 2 | $2^{11}$ | 1 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2914** | 64kB | 2 | 16kB | 4 | 256 | 256 | 32 | 32 | 16 | 40 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{12}$ | 20 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2915** | 64kB | 2 | 32kB | 1 | 50 | 96 | 16 | 32 | 64 | 32 | 3 | $2^{13}$ | 3 | 10 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2916** | 64kB | 2 | 16kB | 2 | 64 | 96 | 32 | 32 | 64 | 64 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2917** | 32kB | 2 | 16kB | 2 | 128 | 128 | 32 | 8 | 64 | 40 | 1 | $2^{10}$ | 1 | 10 | $2^{11}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2918** | 32kB | 4 | 4kB | 2 | 50 | 96 | 64 | 8 | 64 | 32 | 1 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2919** | 32kB | 4 | 16kB | 1 | 50 | 96 | 32 | 8 | 64 | 16 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2920** | 64kB | 4 | 64kB | 4 | 96 | 96 | 32 | 8 | 32 | 128 | 3 | $2^{12}$ | 2 | 11 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2921** | 64kB | 4 | 64kB | 4 | 96 | 96 | 64 | 16 | 8 | 40 | 1 | $2^{14}$ | 3 | 11 | $2^{11}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2922** | 64kB | 2 | 4kB | 2 | 128 | 256 | 64 | 8 | 64 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{2}$ | 3 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2923** | 32kB | 2 | 64kB | 2 | 128 | 128 | 64 | 32 | 16 | 128 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{10}$ | 20 | |
| **#2924** | 64kB | 1 | 8kB | 2 | 128 | 96 | 64 | 16 | 8 | 16 | 1 | $2^{13}$ | 2 | 10 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2925** | 16kB | 2 | 8kB | 2 | 96 | 128 | 32 | 8 | 16 | 32 | 1 | $2^{13}$ | 3 | 10 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2926** | 64kB | 4 | 4kB | 2 | 256 | 128 | 32 | 16 | 32 | 16 | 2 | $2^{14}$ | 3 | 10 | $2^{10}$ | 3 | $2^{14}$ | $2^{10}$ | 18 | |
| **#2927** | 64kB | 2 | 64kB | 1 | 96 | 96 | 16 | 16 | 16 | 128 | 1 | $2^{13}$ | 3 | 11 | $2^{2}$ | 2 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2928** | 32kB | 2 | 4kB | 4 | 64 | 256 | 16 | 16 | 32 | 16 | 1 | $2^{13}$ | 2 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{11}$ | 18 | |
| **#2929** | 32kB | 1 | 8kB | 2 | 96 | 128 | 32 | 16 | 64 | 16 | 3 | $2^{14}$ | 3 | 10 | $2^{10}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2930** | 16kB | 4 | 8kB | 4 | 256 | 128 | 16 | 8 | 16 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 1 | $2^{10}$ | $2^{12}$ | 18 | |
| **#2931** | 64kB | 2 | 32kB | 1 | 50 | 256 | 32 | 8 | 8 | 32 | 1 | $2^{11}$ | 1 | 11 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 20 | |
| **#2932** | 64kB | 4 | 8kB | 4 | 50 | 128 | 32 | 32 | 64 | 32 | 3 | $2^{11}$ | 3 | 12 | $2^{10}$ | 1 | $2^{12}$ | $2^{13}$ | 18 | |
| **#2933** | 64kB | 1 | 32kB | 2 | 256 | 256 | 32 | 16 | 16 | 40 | 2 | $2^{13}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2934** | 32kB | 1 | 8kB | 1 | 128 | 256 | 32 | 16 | 32 | 32 | 2 | $2^{11}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2935** | 64kB | 2 | 32kB | 1 | 128 | 96 | 32 | 8 | 8 | 40 | 2 | $2^{12}$ | 1 | 11 | $2^{11}$ | 1 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2936** | 64kB | 2 | 4kB | 2 | 256 | 128 | 16 | 64 | 64 | 64 | 2 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2937** | 64kB | 4 | 32kB | 4 | 256 | 96 | 16 | 16 | 64 | 128 | 2 | $2^{13}$ | 3 | 12 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2938** | 16kB | 1 | 32kB | 1 | 256 | 96 | 64 | 64 | 32 | 64 | 3 | $2^{12}$ | 1 | 12 | $2^{2}$ | 3 | $2^{10}$ | $2^{13}$ | 18 | |
| **#2939** | 32kB | 1 | 16kB | 4 | 96 | 96 | 64 | 16 | 8 | 16 | 1 | $2^{11}$ | 2 | 11 | $2^{2}$ | 3 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2940** | 64kB | 2 | 4kB | 2 | 64 | 256 | 64 | 8 | 32 | 64 | 3 | $2^{10}$ | 3 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2941** | 32kB | 2 | 4kB | 4 | 256 | 256 | 16 | 16 | 32 | 40 | 1 | $2^{13}$ | 2 | 12 | $2^{2}$ | 2 | $2^{13}$ | $2^{10}$ | 16 | |
| **#2942** | 64kB | 2 | 8kB | 1 | 64 | 128 | 64 | 16 | 64 | 64 | 3 | $2^{11}$ | 1 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{12}$ | 16 | |
| **#2943** | 64kB | 4 | 4kB | 4 | 256 | 128 | 32 | 8 | 8 | 40 | 2 | $2^{10}$ | 3 | 12 | $2^{10}$ | 2 | $2^{11}$ | $2^{13}$ | 18 | |
| **#2944** | 16kB | 1 | 4kB | 4 | 96 | 256 | 32 | 16 | 32 | 16 | 3 | $2^{14}$ | 2 | 10 | $2^{2}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2945** | 16kB | 2 | 32kB | 4 | 96 | 96 | 64 | 16 | 32 | 64 | 2 | $2^{14}$ | 3 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{10}$ | 20 | |
| **#2946** | 16kB | 2 | 64kB | 2 | 256 | 256 | 16 | 64 | 16 | 32 | 2 | $2^{10}$ | 3 | 10 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2947** | 16kB | 4 | 32kB | 1 | 96 | 256 | 64 | 16 | 64 | 128 | 2 | $2^{10}$ | 2 | 10 | $2^{11}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2948** | 16kB | 2 | 64kB | 1 | 64 | 96 | 64 | 32 | 16 | 32 | 1 | $2^{12}$ | 1 | 10 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2949** | 16kB | 4 | 32kB | 2 | 256 | 96 | 16 | 16 | 16 | 16 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2950** | 32kB | 4 | 64kB | 1 | 96 | 256 | 64 | 16 | 64 | 64 | 2 | $2^{13}$ | 1 | 10 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2951** | 32kB | 4 | 8kB | 1 | 50 | 256 | 32 | 16 | 16 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{10}$ | $2^{10}$ | 16 | |
| **#2952** | 16kB | 1 | 16kB | 4 | 50 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{12}$ | 1 | 10 | $2^{10}$ | 1 | $2^{11}$ | $2^{12}$ | 18 | |
| **#2953** | 16kB | 1 | 8kB | 1 | 96 | 256 | 16 | 32 | 64 | 64 | 1 | $2^{11}$ | 1 | 10 | $2^{2}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |
| **#2954** | 16kB | 4 | 4kB | 4 | 128 | 96 | 16 | 64 | 32 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{10}$ | 2 | $2^{13}$ | $2^{11}$ | 18 | |
| **#2955** | 64kB | 4 | 16kB | 4 | 96 | 96 | 64 | 32 | 16 | 128 | 3 | $2^{12}$ | 3 | 12 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 20 | |
| **#2956** | 32kB | 4 | 16kB | 4 | 64 | 256 | 64 | 64 | 64 | 32 | 3 | $2^{11}$ | 1 | 12 | $2^{10}$ | 2 | $2^{14}$ | $2^{12}$ | 20 | |
| **#2957** | 32kB | 1 | 16kB | 2 | 64 | 128 | 16 | 8 | 16 | 128 | 1 | $2^{13}$ | 1 | 12 | $2^{10}$ | 3 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2958** | 64kB | 4 | 64kB | 1 | 96 | 256 | 16 | 64 | 8 | 32 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2959** | 64kB | 4 | 64kB | 4 | 256 | 96 | 16 | 64 | 8 | 128 | 1 | $2^{12}$ | 1 | 11 | $2^{10}$ | 1 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2960** | 16kB | 2 | 16kB | 2 | 128 | 128 | 32 | 32 | 64 | 40 | 2 | $2^{13}$ | 1 | 12 | $2^{11}$ | 3 | $2^{14}$ | $2^{13}$ | 18 | |
| **#2961** | 64kB | 2 | 8kB | 4 | 256 | 256 | 64 | 32 | 16 | 16 | 2 | $2^{12}$ | 1 | 12 | $2^{11}$ | 1 | $2^{12}$ | $2^{10}$ | 18 | |
| **#2962** | 16kB | 1 | 64kB | 1 | 128 | 96 | 32 | 32 | 8 | 64 | 3 | $2^{12}$ | 2 | 11 | $2^{2}$ | 2 | $2^{14}$ | $2^{12}$ | 16 | |
| **#2963** | 16kB | 2 | 4kB | 2 | 96 | 96 | 32 | 64 | 64 | 64 | 3 | $2^{11}$ | 3 | 10 | $2^{2}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2964** | 32kB | 1 | 64kB | 2 | 50 | 128 | 32 | 16 | 32 | 32 | 1 | $2^{12}$ | 1 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2965** | 16kB | 2 | 32kB | 2 | 256 | 128 | 64 | 8 | 8 | 16 | 2 | $2^{14}$ | 2 | 12 | $2^{11}$ | 3 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2966** | 32kB | 2 | 16kB | 4 | 64 | 128 | 32 | 8 | 64 | 128 | 2 | $2^{11}$ | 2 | 12 | $2^{11}$ | 2 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2967** | 32kB | 4 | 8kB | 4 | 96 | 128 | 64 | 32 | 16 | 64 | 3 | $2^{13}$ | 2 | 12 | $2^{10}$ | 2 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2968** | 32kB | 4 | 4kB | 1 | 64 | 256 | 32 | 64 | 32 | 16 | 3 | $2^{10}$ | 3 | 12 | $2^{2}$ | 1 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2969** | 64kB | 1 | 32kB | 4 | 256 | 96 | 64 | 8 | 32 | 64 | 1 | $2^{14}$ | 1 | 10 | $2^{11}$ | 2 | $2^{12}$ | $2^{11}$ | 16 | |
| **#2970** | 16kB | 1 | 8kB | 2 | 64 | 256 | 16 | 16 | 64 | 128 | 3 | $2^{10}$ | 1 | 12 | $2^{11}$ | 3 | $2^{13}$ | $2^{12}$ | 16 | |
| **#2971** | 64kB | 4 | 64kB | 1 | 128 | 128 | 64 | 16 | 64 | 16 | 2 | $2^{14}$ | 1 | 10 | $2^{10}$ | 1 | $2^{13}$ | $2^{13}$ | 16 | |
| **#2972** | 64kB | 4 | 16kB | 2 | 96 | 96 | 16 | 32 | 16 | 40 | 3 | $2^{14}$ | 1 | 12 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 18 | |
| **#2973** | 32kB | 1 | 4kB | 1 | 128 | 96 | 32 | 64 | 64 | 32 | 1 | $2^{13}$ | 2 | 11 | $2^{2}$ | 1 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2974** | 32kB | 4 | 32kB | 4 | 50 | 256 | 32 | 64 | 64 | 128 | 1 | $2^{12}$ | 3 | 11 | $2^{2}$ | 2 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2975** | 64kB | 1 | 32kB | 2 | 64 | 96 | 64 | 8 | 8 | 128 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 2 | $2^{14}$ | $2^{13}$ | 20 | |
| **#2976** | 16kB | 4 | 8kB | 2 | 128 | 256 | 32 | 32 | 8 | 64 | 3 | $2^{14}$ | 3 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{13}$ | 20 | |
| **#2977** | 32kB | 1 | 64kB | 4 | 128 | 128 | 64 | 8 | 8 | 40 | 3 | $2^{10}$ | 2 | 11 | $2^{10}$ | 3 | $2^{13}$ | $2^{11}$ | 20 | |
| **#2978** | 64kB | 1 | 32kB | 1 | 50 | 96 | 32 | 8 | 16 | 64 | 3 | $2^{13}$ | 2 | 11 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2979** | 64kB | 2 | 64kB | 4 | 128 | 96 | 16 | 64 | 32 | 32 | 2 | $2^{12}$ | 2 | 11 | $2^{11}$ | 3 | $2^{10}$ | $2^{13}$ | 16 | |
| **#2980** | 16kB | 2 | 32kB | 1 | 256 | 128 | 16 | 16 | 32 | 40 | 2 | $2^{14}$ | 2 | 11 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2981** | 64kB | 4 | 64kB | 2 | 96 | 96 | 32 | 32 | 64 | 32 | 1 | $2^{10}$ | 2 | 10 | $2^{11}$ | 3 | $2^{10}$ | $2^{12}$ | 16 | |

Continued on next page.

| Core | DS | DW | IS | IW | IR | FPR | IQ | LQ | SQ | ROB | GCB | GE | LCB | LHB | LHE | CCB | CE | BE | BT | Page |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **#2982** | 64kB | 4 | 8kB | 4 | 50 | 256 | 16 | 64 | 8 | 64 | 1 | $2^{10}$ | 1 | 12 | $2^{2}$ | 3 | $2^{14}$ | $2^{11}$ | 18 | |
| **#2983** | 32kB | 2 | 8kB | 2 | 96 | 256 | 32 | 16 | 16 | 40 | 1 | $2^{11}$ | 1 | 12 | $2^{10}$ | 3 | $2^{10}$ | $2^{11}$ | 20 | |
| **#2984** | 32kB | 2 | 16kB | 4 | 256 | 256 | 16 | 16 | 16 | 40 | 3 | $2^{13}$ | 1 | 11 | $2^{10}$ | 1 | $2^{13}$ | $2^{10}$ | 18 | |
| **#2985** | 16kB | 2 | 64kB | 1 | 64 | 128 | 16 | 16 | 8 | 16 | 3 | $2^{12}$ | 2 | 12 | $2^{2}$ | 1 | $2^{10}$ | $2^{11}$ | 16 | |
| **#2986** | 16kB | 4 | 32kB | 4 | 128 | 96 | 32 | 64 | 8 | 32 | 2 | $2^{11}$ | 2 | 12 | $2^{10}$ | 3 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2987** | 64kB | 1 | 16kB | 1 | 50 | 128 | 32 | 32 | 16 | 40 | 2 | $2^{11}$ | 3 | 10 | $2^{11}$ | 3 | $2^{11}$ | $2^{10}$ | 20 | |
| **#2988** | 32kB | 4 | 8kB | 1 | 50 | 96 | 64 | 8 | 16 | 32 | 3 | $2^{11}$ | 3 | 10 | $2^{11}$ | 2 | $2^{14}$ | $2^{10}$ | 16 | |
| **#2989** | 64kB | 2 | 4kB | 1 | 256 | 96 | 16 | 16 | 64 | 128 | 1 | $2^{11}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{11}$ | 16 | |
| **#2990** | 64kB | 2 | 16kB | 2 | 256 | 256 | 32 | 8 | 64 | 32 | 1 | $2^{14}$ | 2 | 11 | $2^{2}$ | 2 | $2^{12}$ | $2^{12}$ | 20 | |
| **#2991** | 32kB | 1 | 32kB | 4 | 64 | 128 | 64 | 8 | 16 | 40 | 1 | $2^{10}$ | 3 | 11 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2992** | 32kB | 2 | 8kB | 2 | 96 | 96 | 32 | 64 | 64 | 64 | 1 | $2^{11}$ | 1 | 12 | $2^{10}$ | 1 | $2^{10}$ | $2^{13}$ | 20 | |
| **#2993** | 32kB | 1 | 32kB | 1 | 64 | 96 | 16 | 64 | 32 | 64 | 3 | $2^{13}$ | 1 | 12 | $2^{10}$ | 2 | $2^{12}$ | $2^{10}$ | 16 | |
| **#2994** | 16kB | 1 | 4kB | 2 | 64 | 96 | 64 | 32 | 8 | 16 | 1 | $2^{10}$ | 2 | 12 | $2^{11}$ | 2 | $2^{10}$ | $2^{12}$ | 20 | |
| **#2995** | 16kB | 2 | 64kB | 1 | 50 | 96 | 64 | 8 | 16 | 16 | 3 | $2^{13}$ | 3 | 12 | $2^{2}$ | 2 | $2^{11}$ | $2^{11}$ | 16 | |
| **#2996** | 64kB | 4 | 4kB | 2 | 64 | 256 | 16 | 32 | 64 | 16 | 1 | $2^{13}$ | 2 | 11 | $2^{10}$ | 2 | $2^{14}$ | $2^{13}$ | 16 | |
| **#2997** | 64kB | 4 | 32kB | 1 | 256 | 128 | 64 | 64 | 8 | 64 | 3 | $2^{14}$ | 1 | 11 | $2^{11}$ | 3 | $2^{13}$ | $2^{11}$ | 16 | |
| **#2998** | 32kB | 4 | 8kB | 2 | 96 | 96 | 64 | 32 | 32 | 16 | 1 | $2^{14}$ | 3 | 12 | $2^{11}$ | 2 | $2^{13}$ | $2^{13}$ | 18 | |
| **#2999** | 64kB | 1 | 8kB | 4 | 256 | 128 | 64 | 32 | 8 | 128 | 3 | $2^{12}$ | 3 | 11 | $2^{11}$ | 1 | $2^{13}$ | $2^{11}$ | 20 | |
| **#3000** | 64kB | 4 | 8kB | 1 | 128 | 128 | 64 | 32 | 64 | 32 | 1 | $2^{12}$ | 1 | 12 | $2^{2}$ | 1 | $2^{14}$ | $2^{12}$ | 18 | |

# Bibliography


[1] Margareta Ackerman and Shai Ben-David. Clusterability: A theoretical study. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009. Available: http://www.jmlr.org/proceedings/papers/v5/ackerman09a.html.

[2] Salem F. Adra and Peter J. Fleming. Diversity management in evolutionary many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 15(2):183–195, Apr. 2011. DOI: 10.1109/TEVC.2010.2058117.

[3] Massimo Alioto, Elio Consoli, and Gaetano Palumbo. From energy-delay metrics to constraints on the design of digital circuits. *International Journal of Circuit Theory and Applications*, 40(8):815–834, 2012. DOI: 10.1002/cta.757.

[4] Mohamad Hammam Alsafrjalani and Ann Gordon-Ross. Dynamic scheduling for reduced energy in configuration-subsetted heterogeneous multicore systems. In *Proceedings of the 12th International Conference on Embedded and Ubiquitous Computing (EUC)*. IEEE, Aug. 2014. DOI: 10.1109/EUC.2014.12.

[5] Murali Annavaram, Ed Grochowski, and John Shen. Mitigating Amdahl's Law through EPI throttling. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, 2005. DOI: 10.1109/ISCA.2005.36.

[6] Manish Arora, Srilatha Manne, Yasuko Eckert, Indrani Paul, Nuwan Jayasena, and Dean Tullsen. A comparison of core power gating strategies implemented in modern hardware. In *Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2014. DOI: 10.1145/2591971.2592017.

[7] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. Energy-performance tradeoffs in processor architecture and circuit design: A marginal cost analysis. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2010. DOI: 10.1145/1815961.1815967.

[8] R. Iris Bahar and Srilatha Manne. Power and energy reduction via pipeline balancing. In *Proceedings of the 28th International Symposium on Computer Architecture (ISCA)*, 2001. DOI: 10.1109/ISCA.2001.937451.

[9] Saisanthosh Balakrishnan, Ravi Rajwar, Mike Upton, and Konrad Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA)*, Jun. 2005. DOI: 10.1109/ISCA.2005.51.

[10] Lars Bauer, Muhammad Shafique, Simon Kramer, and Jörg Henkel. RISPP: Rotating instruction set processing platform. In *Proceedings of the 44th Design Automation Conference (DAC)*. ACM, 2007. DOI: 10.1145/1278480.1278678.

[11] Brad D. Bingham and Mark R. Greenstreet. Computation with energy-time trade-offs: Models, algorithms and lower-bounds. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, Dec. 2008. DOI: 10.1109/ISPA.2008.127.

[12] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, Aug. 2011. DOI: 10.1145/2024716.2024718.

[13] Ramazan Bitirgen, Engin İpek, and José Martínez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st International Symposium on Microarchitecture (MICRO)*. IEEE, Nov. 2008. DOI: 10.1109/MICRO.2008.4771801.

[14] Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2013. DOI: 10.1109/HPCA.2013.6522302.

[15] Emily R. Blem, Hadi Esmaeilzadeh, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Multicore model from abstract single core inputs. *Computer Architecture Letters*, 12(2):59–62, Jul. 2013. DOI: 10.1109/L-CA.2012.27.

[16] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011. DOI: 10.1145/1941487.1941507.

[17] Maximilien B. Breughe and Lieven Eeckhout. Selecting representative benchmark inputs for exploring microprocessor design spaces. *ACM Transactions Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013. DOI: 10.1145/2555289.2555294.

[18] David M. Brooks, Pradip Bose, Stanley Schuster, Hans M. Jacobson, Prabhakar Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor V. Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and

modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6): 26–44, Nov. 2000. DOI: 10.1109/40.888701.

[19] David M. Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, Jun. 2000. DOI: 10.1109/ISCA.2000.854380.

[20] Thomas D. Burd and Robert W. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2000. DOI: 10.1145/344166.344181.

[21] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Michael Dahlin, Lizy Kurian John, Calvin Lin, Charles R. Moore, James H. Burrill, Robert G. McDonald, and William Yode. Scaling to the end of silicon with EDGE architectures. *IEEE Computer*, 37(7):44–55, Jul. 2004. DOI: 10.1109/MC.2004.65.

[22] Graeme Burton. AMD to design new micro-architecture for 2015 launch under chip guru Jim Keller, May 2014. Available: http://www.computing.co.uk/ctg/news/2344223/amd-to-design-new-micro-architecture-for-2015-launch-under-chip-guru-jim-keller. Accessed 2016-02-29.

[23] Andrew Cassidy, Kai Yu, Haolang Zhou, and Andreas G. Andreou. A high-level analytical model for application specific CMP design exploration. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*, Mar. 2011. DOI: 10.1109/DATE.2011.5763180.

[24] Juan M. Cebrián, Juan L. Aragón, José M. Garcia, Pavlos Petoumenos, and Stefanos Kaxiras. Efficient microarchitecture policies for accurately adapting to power constraints. In *International Symposium on Parallel Distributed Processing (IPDPS)*. IEEE, May 2009. DOI: 10.1109/IPDPS.2009.5161022.

[25] Geoffrey Challen and Mark Hempstead. The case for power-agile computing. In *Proceedings of the 13th USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, May 2011. Available: https://www.usenix.org/conference/hotosxiii/case-power-agile-computing.

[26] Kiran Chandramohan and Michael F. P. O'Boyle. Partitioning data-parallel programs for heterogeneous MPSoCs: Time and energy design space exploration. In *Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM, 2014. DOI: 10.1145/2597809.2597822.

[27] Kiran Chandramohan and Michael F. P. O'Boyle. A compiler framework for automatically mapping data parallel programs to heterogeneous MPSoCs. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. ACM, 2014. DOI: 10.1145/2656106.2656107.

[28] Jie Chen, Guru Venkataramani, and Gabriel Parmer. The need for power debugging in the multi-core environment. *Computer Architecture Letters*, 11 (4):57–60, Jul. 2012. DOI: 10.1109/L-CA.2012.1.

[29] Jie Chen, Fan Yao, and Guru Venkataramani. Watts-inside: A hardware-software cooperative approach for multicore power debugging. In *Proceedings of the 31st International Conference on Computer Design (ICCD)*. IEEE, Oct. 2013. DOI: 10.1109/ICCD.2013.6657062.

[30] Andrew A. Chien, Allan Snavely, and Mark Gahagan. 10x10: A general-purpose architectural approach to heterogeneity and energy efficiency. *Procedia Computer Science*, 4:1987–1996, 2011. DOI: 10.1016/j.procs.2011.04.217.

[31] Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *International Conference on Computer Aided Design (ICCAD)*, 2004. DOI: 10.1109/ICCAD.2004.1382538.

[32] Niket Kumar Choudhary, Salil V. Wadhavkar, Tanmay A. Shah, Hiran Mayukh, Jayneel Gandhi, Brandon H. Dwiel, Sandeep Navada, Hashem Hashemi Najafabadi, and Eric Rotenberg. FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2011. DOI: 10.1145/2000064.2000067.

[33] Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? In *Proceedings of the 43rd International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2010. DOI: 10.1109/MICRO.2010.36.

[34] Patrick Cogez, Mart Graef, Bert Huizing, Reinhard Mahnkopf, Hidemi Ishiuchi, Junji Shindo, Siyoung Choi, JaeSung Roh, Carlos H. Diaz, Burn Lin, Bob Doering, Paolo Gargini, and Ian Steff. *International technology roadmap for semiconductors, 2011 Edition*, chapter Process Integration, Devices, and Structures. ITRS, 2011.

[35] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, and Glenn Reinman. CHARM: A composable heterogeneous accelerator-rich microprocessor. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2012. DOI: 10.1145/2333660.2333747.

[36] W. J. Conover. *Practical nonparametric statistics.* Wiley series in probability and statistics: Applied probability and statistics section. John Wiley & Sons, Inc., 3rd edition, 1999.

[37] Charlie Curtsinger and Emery D. Berger. STABILIZER: Statistically sound performance evaluation. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2013. DOI: 10.1145/2451116.2451141.

[38] Kenneth Czechowski, Victor W. Lee, Ed Grochowski, Ronny Ronen, Ronak Singhal, Richard Vuduc, and Pradeep Dubey. Improving the energy efficiency of big cores. In *Proceedings of the 41st International Symposium on Computer Architecture (ISCA)*. IEEE, June 2014. DOI: 10.1109/ISCA.2014.6853219.

[39] Michael Dales. Managing a reconfigurable processor in a general purpose workstation environment. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*, 2003. DOI: 10.1109/DATE.2003.1253732.

[40] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, Apr. 2002. DOI: 10.1109/4235.996017.

[41] Rajagopalan Desikan, Simha Sethumadhavan, Doug Burger, and Stephen W. Keckler. Scalable selective re-execution for EDGE architectures. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, (ASPLOS)*, 2004. DOI: 10.1145/1024393.1024408.

[42] Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. Execution migration in a heterogeneous-ISA chip multiprocessor. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2012. DOI: 10.1145/2150976.2151004.

[43] Gaurav Dhiman, Kishore Kumar Pusukuri, and Tajana Rosing. Analysis of dynamic voltage scaling for system level energy management. In *Proceedings of the USENIX Workshop on Power Aware Computing Systems (HotPower)*, 2008. Available: http://dl.acm.org/citation.cfm?id=1855610.1855619.

[44] Ashutosh S. Dhodapkar and James E. Smith. Comparing program phase detection techniques. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2003. DOI: 10.1109/MICRO.2003.1253197.

[45] Yang Ding, Mahmut Kandemir, Padma Raghavan, and Mary Jane Irwin. A helper thread based EDP reduction scheme for adapting application execution in CMPs. In *Proceedings of the International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, Apr. 2008. DOI: 10.1109/IPDPS.2008.4536297.

[46] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 43rd International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2010. DOI: 10.1109/MICRO.2010.14.

[47] Rotem Efraim, Ran Ginosar, Uri Weiser, and Avi Mendelson. Energy aware race to halt: A down to EARtH approach for platform energy management. *Computer Architecture Letters*, 13(1):25–28, Jan. 2014. ISSN 1556-6056. DOI: 10.1109/L-CA.2012.32.

[48] Aristides Efthymiou and Jim D. Garside. Adaptive pipeline depth control for processor power-management. In *Proceedings of the 20th International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*. IEEE, 2002. DOI: 10.1109/ICCD.2002.1106812.

[49] David Eklov, David Black-Schaffer, and Erik Hagersten. StatCC: A statistical cache contention model. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2010. DOI: 10.1145/1854273.1854347.

[50] Hadi Esmaeilzadeh, Emily Blem, Reneé St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*, Jun. 2011. DOI: 10.1145/2000064.2000108.

[51] Maja Etinski, Julita Corbalán, Jesús Labarta, and Mateo Valero. Understanding the future of energy-performance trade-off via DVFS in HPC environments. *Journal of Parallel and Distributed Computing*, 72(4):579–590, 2012. DOI: 10.1016/j.jpdc.2012.01.006.

[52] Stijn Eyerman and Lieven Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, May 2008. DOI: 10.1109/MM.2008.44.

[53] Stijn Eyerman and Lieven Eeckhout. The benefit of SMT in the multi-core era: Flexibility towards degrees of thread-level parallelism. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2014. DOI: 10.1145/2541940.2541954.

[54] Stijn Eyerman, Kenneth Hoste, and Lieven Eeckhout. Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2011. DOI: 10.1109/ISPASS.2011.5762738.

[55] Stijn Eyerman, Pierre Michaud, and Wouter Rogiest. Multiprogram throughput metrics: A systematic approach. *ACM Transactions on Architecture and Code Optimization (TACO)*, 11(3), Oct. 2014. DOI: 10.1145/2663346.

[56] Ali Farhang-Mehr and Shapour Azarm. Diversity assessment of Pareto optimal solution sets: An entropy approach. In *Proceedings of the Congress on Evolutionary Computation (CEC)*. IEEE, May 2002. DOI: 10.1109/CEC.2002.1007015.

[57] Amin Farmahini-Farahani, Nam Sung Kim, and Katherine Morrow. Energy-efficient reconfigurable cache architectures for accelerator-enabled embedded systems. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2014. DOI: 10.1109/ISPASS.2014.6844485.

[58] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *Proceedings of the 29th International Symposium on Computer Architecture (ISCA)*, May 2002. DOI: 10.1109/ISCA.2002.1003572.

[59] Vincent W. Freeh, David K. Lowenthal, Feng Pan, Nandini Kappiah, Rob Springer, Barry Rountree, and Mark E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, Jun. 2007. DOI: 10.1109/TPDS.2007.1026.

[60] Peter Gavin, David Whalley, and Magnus Själander. Reducing instruction fetch energy in multi-issue processors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013. DOI: 10.1145/2541228.2555318.

[61] Mark Gebhart, Bertrand A. Maher, Katherine E. Coons, Jeffrey R. Diamond, Paul Gratz, Mario Marino, Nitya Ranganathan, Behnam Robatmili, Aaron Smith, James H. Burrill, Stephen W. Keckler, Doug Burger, and Kathryn S. McKinley. An evaluation of the TRIPS computer system. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2009. DOI: 10.1145/1508244.1508246.

[62] Catherine H. Gebotys and Robert J. Gebotys. Power minimization in heterogeneous processing. In *Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS)*. IEEE, Jan. 1996. DOI: 10.1109/HICSS.1996.495478.

[63] Giorgis Georgakoudis, Dimitrios S. Nikolopoulos, Hans Vandierendonck, and Spyros Lalis. Fast dynamic binary rewriting for flexible thread migration on shared-ISA heterogeneous MPSoCs. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, Jul. 2014. DOI: 10.1109/SAMOS.2014.6893207.

[64] Tony Givargis, Frank Vahid, and Jörg Henkel. System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE, Nov. 2001. DOI: 10.1109/ICCAD.2001.968593.

[65] Diana Göhringer, Michael Hübner, Volker Schatz, and Jürgen Becker. Runtime adaptive multi-processor system-on-chip: RAMPSoC. In *Proceedings of the International Symposium on Parallel and Distributed Processing (IPDPS)*. IEEE, Apr. 2008. DOI: 10.1109/IPDPS.2008.4536503.

[66] Ricardo Gonzalez and Mark Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sep. 1996. DOI: 10.1109/4.535411.

[67] Cecilia González-Álvarez, Jennifer B. Sartor, Carlos Álvarez, Daniel Jiménez-González, and Lieven Eeckhout. Accelerating an application domain with specialized functional units. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013. DOI: 10.1145/2555289.2555303.

[68] Nathan Goulding-Hotta, Jack Sampson, Ganesh Venkatesh, Saturnino Garcia, Joe Auricchio, Po-Chao Huang, Manish Arora, Siddhartha Nath, Vikram Bhatt, Jonathan Babb, Steven Swanson, and Michael Bedford Taylor. The GreenDroid mobile application processor: An architecture for silicon's dark future. *IEEE Micro*, 31(2):86–95, Mar. 2011. DOI: 10.1109/MM.2011.18.

[69] Peter Greenhalgh. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. White paper, ARM Ltd., Sep. 2011.

[70] Ed Grochowski, Ronny Ronen, John Shen, and Hong Wang. Best of both latency and throughput. In *Proceedings of the International Conference on Computer Design (ICCD)*. IEEE, 2004. Available: http://dl.acm.org/citation.cfm?id=1032648.1033367.

[71] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. Strategies for anticipating risk in heterogeneous system design. In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2014. DOI: 10.1109/HPCA.2014.6835926.

[72] Apala Guha, Pietro Cicotti Allan, and Andrew A. Chien. The 10x10 foundation for heterogeneity: Clustering applications by computation and memory behavior. Technical report, University of Chicago, Jan. 2012. Available: http://newtraell.cs.uchicago.edu/research/publications/techreports/TR-2012-01.

[73] Apala Guha, Yao Zhang, Raihan ur Rasool, and Andrew A. Chien. Systematic evaluation of workload clustering for extremely energy-efficient architectures. *ACM SIGARCH Computer Architecture News*, 41(2):22–29, May 2013. DOI: 10.1145/2490302.2490307.

[74] Anshuman Gupta, Jack Sampson, and Michael Bedford Taylor. Quality time: A simple online technique for quantifying multicore execution efficiency. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2014. DOI: 10.1109/ISPASS.2014.6844481.

[75] Shantanu Gupta, Shuguang Feng, Amin Ansari, Ganesh Dasika, and Scott Mahlke. CoreGenesis: Erasing core boundaries for robust and configurable performance. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2010. DOI: 10.1145/1854273.1854357.

[76] Shantanu Gupta, Shuguang Feng, Amin Ansari, and Scott Mahlke. StageNet: A reconfigurable fabric for constructing dependable CMPs. *IEEE Transactions on Computers*, 60(1):5–19, Jan. 2011. DOI: 10.1109/TC.2010.205.

[77] Anthony Gutierrez, Ronald G. Dreslinski, Thomas F. Wenisch, Trevor N. Mudge, Ali G. Saidi, Christopher D. Emmons, and Nigel C. Paver. Full-system analysis and characterization of interactive smartphone applications. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*. IEEE, Nov. 2011. DOI: 10.1109/IISWC.2011.6114205.

[78] Anthony Gutierrez, Joseph Pusdesris, Ronald G. Dreslinski, Trevor N. Mudge, Chander Sudanthi, Christopher D. Emmons, Mitchell Hayenga, and Nigel C. Paver. Sources of error in full-system simulation. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2014. DOI: 10.1109/ISPASS.2014.6844457.

[79] John Hennessy, Daniel Citron, David Patterson, and Gurindar Sohi. The use and abuse of SPEC: An ISCA panel. *IEEE Micro*, 23(4):73–77, Jul. 2003. DOI: 10.1109/MM.2003.1225977.

[80] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 41(7):33–38, Jul. 2008. DOI: 10.1109/MC.2008.209.

[81] Houman Homayoun, Shahin Golshan, Eli Bozorgzadeh, Alexander Veidenbaum, and Fadi J. Kurdahi. On leakage power optimization in clock tree networks for ASICs and general-purpose processors. *Sustainable Computing: Informatics and Systems*, 1(1):75–87, 2011. DOI: 10.1016/j.suscom.2010.10.005.

[82] Kenneth Hoste and Lieven Eeckhout. Comparing benchmarks using key microarchitecture-independent characteristics. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*. IEEE, Oct. 2006. DOI: 10.1109/IISWC.2006.302732.

[83] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy Kurian John, and Koen De Bosschere. Performance prediction based on inherent program similarity. In *Proceedings of the 15$^{th}$ International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2006. DOI: 10.1145/1152154.1152174.

[84] Michael C. Huang, Jose Renau, and Josep Torrellas. Positional adaptation of processors: Application to energy reduction. In *Proceedings of the 30$^{th}$ International Symposium on Computer Architecture (ISCA)*. IEEE, Jun. 2003. DOI: 10.1109/ISCA.2003.1206997.

[85] Engin İpek, Meyrem Kırman, Nevin Kırman, and José Martínez. Core fusion: Accommodating software diversity in chip multiprocessors. In *Proceedings of the 34$^{th}$ International Symposium on Computer Architecture (ISCA)*. ACM, 2007. DOI: 10.1145/1250662.1250686.

[86] Chuntao Jiang, Zhibin Yu, Hai Jin, Cheng-Zhong Xu, Lieven Eeckhout, Wim Heirman, Trevor E. Carlson, and Xiaofei Liao. PCantorSim: Accelerating parallel architecture simulation through fractal-based sampling. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013. DOI: 10.1145/2541228.2555305.

[87] Sukhun Kang and Rakesh Kumar. Magellan: A search and machine learning-based framework for fast multi-core design space exploration and optimization. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*. ACM, 2008. DOI: 10.1145/1403375.1403721.

[88] Tejas S. Karkhanis and James E. Smith. Automated design of application specific superscalar processors: An analytical approach. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*. ACM, 2007. DOI: 10.1145/1250662.1250712.

[89] Cansu Kaynak, Boris Grot, and Babak Falsafi. Shift: Shared history instruction fetch for lean-core server processors. In *Proceedings of the 46th International Symposium on Microarchitecture (MICRO)*. ACM, 2013. DOI: 10.1145/2540708.2540732.

[90] Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. GPUs and the future of parallel computing. *IEEE Micro*, 31(5): 7–17, Sep. 2011. DOI: 10.1109/MM.2011.89.

[91] Changkyu Kim, Simha Sethumadhavan, M.S. Govindan, Nitya Ranganathan, Divya Gulati, Doug Burger, and Stephen W. Keckler. Composable lightweight processors. In *Proceedings of the 40th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2007. DOI: 10.1109/MICRO.2007.41.

[92] Ilhyun Kim and Mikko H. Lipasti. Understanding scheduling replay schemes. In *Proceedings of the 10th International Conference on High-Performance Computer Architecture (HPCA)*, Feb. 2004. DOI: 10.1109/HPCA.2004.10011.

[93] Wonyoung Kim, Meeta S. Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2008. DOI: 10.1109/HPCA.2008.4658633.

[94] Joshua D. Knowles and David Corne. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation*, 8(2): 149–172, 2000. DOI: 10.1162/106365600568167.

[95] Akash Kumar, Andreas Hansson, Jos Huisken, and Henk Corporaal. An FPGA design flow for reconfigurable network-based multi-processor systems on chip. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*, Apr. 2007. DOI: 10.1109/DATE.2007.364577.

[96] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2003. DOI: 10.1109/MICRO.2003.1253185.

[97] Rakesh Kumar, Norman P. Jouppi, and Dean M. Tullsen. Conjoined-core chip multiprocessing. In *Proceedings of the 37th Annual International Symposium on Microarchitecture (MICRO)*, Dec. 2004. DOI: 10.1109/MICRO.2004.12.

[98] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA)*, Jun. 2004. DOI: 10.1109/ISCA.2004.1310764.

[99] Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2006. DOI: 10.1145/1152154.1152162.

[100] Frank Kursawe. A variant of evolution strategies for vector optimization. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*. Springer Berlin Heidelberg, 1991. DOI: 10.1007/BFb0029752.

[101] Vahid Lari, Shravan Muddasani, Srinivas Boppu, Frank Hannig, and Jürgen Teich. Design of low power on-chip processor arrays. In *Proceedings of the 23rd International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*. IEEE, July 2012. DOI: 10.1109/ASAP.2012.10.

[102] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation*, 10(3), 2002. DOI: 10.1162/106365602760234108.

[103] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Workshop on Power Aware Computing and Systems (HotPower)*, Oct. 2010. Available: https://www.usenix.org/legacy/events/hotpower10/tech/full_papers/LeSueur.pdf.

[104] Etienne Le Sueur and Gernot Heiser. Slow down or sleep, that is the question. In *Proceedings of the USENIX Annual Technical Conference (USENIXATC)*. USENIX Association, 2011. Available: http://dl.acm.org/citation.cfm?id=2002181.2002197.

[105] Benjamin C. Lee and David Brooks. Roughness of microarchitectural design topologies and its implications for optimization. In *Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2008. DOI: 10.1109/HPCA.2008.4658643.

[106] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2006. DOI: 10.1145/1168857.1168881.

[107] Benjamin C. Lee and David M. Brooks. Illustrative design space studies with microarchitectural regression models. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2007. DOI: 10.1109/HPCA.2007.346211.

[108] Jaewon Lee, Hanhwi Jang, and Jangwoo Kim. RpStacks: Fast and accurate processor design space exploration using representative stall-event stacks. In *Proceedings of the 47th International Symposium on Microarchitecture (MICRO)*. ACM, Dec. 2014. DOI: 10.1109/MICRO.2014.26.

[109] Rainer Leupers, Lieven Eeckhout, Grant Martin, Frank Schirrmeister, Nigel P. Topham, and Xiaotao Chen. Virtual manycore platforms: Moving towards 100+ processor cores. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*, Mar. 2011. DOI: 10.1109/DATE.2011.5763121.

[110] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture (MICRO)*. ACM, Dec. 2009. DOI: 10.1145/1669112.1669172.

[111] Tong Li, Paul Brett, Rob Knauerhase, David Koufaty, Dheeraj Reddy, and Scott Hahn. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *Proceedings of the 16th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Jan. 2010. DOI: 10.1109/HPCA.2010.5416660.

[112] Hung-Yi Liu, Ilias Diakonikolas, Michele Petracca, and Luca Carloni. Supervised design space exploration by compositional approximation of Pareto sets. In *Proceedings of the 48th Design Automation Conference (DAC)*. ACM, 2011. DOI: 10.1145/2024724.2024818.

[113] Qixiao Liu, Miquel Moreto, Victor Jimenez, Jaume Abella, Francisco J. Cazorla, and Mateo Valero. Hardware support for accurate per-task energy metering in multicore systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), Dec. 2013. DOI: 10.1145/2541228.2555291.

[114] Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Faissal M. Sleiman, Ronald Dreslinski, Thomas F. Wenisch, and Scott Mahlke. Composite cores: Pushing heterogeneity into a core. In *Proceedings of the 45th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2012. DOI: 10.1109/MICRO.2012.37.

[115] Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Ronald Dreslinski Jr., Thomas F. Wenisch, and Scott Mahlke. Heterogeneous microarchitectures trump voltage scaling for low-power cores. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, Aug. 2014. DOI: 10.1145/2628071.2628078.

[116] Kun Luo, Jayanth Gummaraju, and Manoj Franklin. Balancing thoughput and fairness in SMT processors. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2001. DOI: 10.1109/ISPASS.2001.990695.

[117] Diana Marculescu and Anoop Iyer. Application-driven processor design exploration for power-performance trade-off analysis. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. IEEE, Nov. 2001. DOI: 10.1109/ICCAD.2001.968638.

[118] Giovanni Mariani, Prabhat Avasare, Geert Vanmeerbeeck, Chantal Ykman-Couvreur, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE)*, Mar. 2010. DOI: 10.1109/DATE.2010.5457211.

[119] Alain J. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 77(2–4):181–187, 2001. DOI: 10.1016/S0020-0190(00)00214-3.

[120] Alain J. Martin, Andrew Lines, Rajit Manohar, Mika Nyström, Paul I. Pénzes, Robert Southworth, and Uri Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Proceedings of the 17$^{th}$ Conference on Advanced Research in VLSI (ARVLSI)*, Sep. 1997. DOI: 10.1109/ARVLSI.1997.634853.

[121] Alain J. Martin, Mika Nyström, and Paul I. Pénzes. Et$^2$: A metric for time and energy efficiency of computation. In Robert Graybill and Rami Melhem, editors, *Power Aware Computing*, Series in Computer Science, pages 293–315. Springer, 2002. DOI: 10.1007/978-1-4757-6217-4.

[122] Daniel S. McFarlin, Charles Tucker, and Craig Zilles. Discerning the dominant out-of-order performance advantage: Is it speculation or dynamism? In *Proceedings of the 18$^{th}$ International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2013. DOI: 10.1145/2451116.2451143.

[123] MediaTek Inc. MediaTek launches the MediaTek Helio X20, May 2015. Available: http://www.mediatek.com/en/news-events/mediatek-news/mediatek-launches-the-mediatek-helio-x20-the-worlds-first-mobile-soc-featuring-tri-cluster-cpu-architecture/. Accessed 2015-05-18.

[124] Nandish Mehta and Bharadwaj Amrutur. Dynamic supply and threshold voltage scaling for CMOS digital circuits using in-situ power monitor. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(5):892–901, May 2012. DOI: 10.1109/TVLSI.2011.2132765.

[125] Pierre Michaud. Demystifying multicore throughput metrics. *Computer Architecture Letters*, 12(2):63–66, Jul. 2013. DOI: 10.1109/L-CA.2012.25.

[126] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. A case for heterogeneous on-chip interconnects for CMPs. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2011. DOI: 10.1145/2000064.2000111.

[127] Tomer Y. Morad, Uri C. Weiser, Avinoam Kolodny, Mateo Valero, and Eduard Ayguadé. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *Computer Architecture Letters*, 5(1):14–17, Jan. 2006. DOI: 10.1109/L-CA.2006.6.

[128] Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt. Runahead execution: An alternative to very large instruction windows for out-of-order processors. In *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2003. DOI: 10.1109/HPCA.2003.1183532.

[129] Nachiappan Chidambaram Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Mahmut T. Kandemir, Anand Sivasubramaniam, and Chita R. Das. GemDroid: A framework to evaluate mobile platforms. In *Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2014. DOI: 10.1145/2591971.2591973.

[130] Ramadass Nagarajan, Karthikeyan Sankaralingam, Doug Burger, and Stephen W. Keckler. A design space evaluation of grid processor architectures. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO)*, Dec. 2001. DOI: 10.1109/MICRO.2001.991104.

[131] Hashem H. Najaf-abadi and Eric Rotenberg. Configurational workload characterization. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, April 2008. DOI: 10.1109/ISPASS.2008.4510747.

[132] Hashem H. Najaf-abadi and Eric Rotenberg. Architectural contesting. In *Proceedings of the 15th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2009. DOI: 10.1109/HPCA.2009.4798254.

[133] Hashem H. Najaf-abadi, Niket K. Choudhary, and Eric Rotenberg. Core-selectability in chip multiprocessors. In *Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2009. DOI: 10.1109/PACT.2009.44.

[134] Sandeep Navada, Niket K. Choudhary, Salil V. Wadhavkar, and Eric Rotenberg. A unified view of non-monotonic core selection and application steering in heterogeneous chip multiprocessors. In *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2013. Available: http://dl.acm.org/citation.cfm?id=2523721.2523743.

[135] Siddharth Nilakantan, Steven J. Battle, and Mark Hempstead. Metrics for early-stage modeling of many-accelerator architectures. *Computer Architecture Letters*, 12(1):25–28, Jan. 2013. DOI: 10.1109/L-CA.2012.9.

[136] NVIDA Corporation. Tegra 3 multi-core processors, n.d. Available: http://www.nvidia.com/object/tegra-3-processor.html. Accessed 2014-02-17.

[137] Santiago Pagani, Heba Khdr, Waqaas Munawar, Jian-Jia Chen, Muhammad Shafique, Minming Li, and Jörg Henkel. TSP: Thermal safe power—Efficient power budgeting for many-core systems in dark silicon. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct. 2014. DOI: 10.1145/2656075.2656103.

[138] Sankaralingam Panneerselvam and Michael M. Swift. Chameleon: Operating system support for dynamic processors. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2012. DOI: 10.1145/2150976.2150988.

[139] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSS: A full system simulator for multicore x86 CPUs. In *Proceedings of the 48th Design Automation Conference (DAC)*. ACM, 2011. DOI: 10.1145/2024724.2024954.

[140] Nam Khanh Pham, Amit Kumar Singh, Akash Kumar, and Khin Mi Mi Aung. Incorporating energy and throughput awareness in design space exploration and run-time mapping for heterogeneous MPSoCs. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, Sep. 2013. DOI: 10.1109/DSD.2013.61.

[141] Jason A. Poovey, Markus Levy, Shay Gal-On, and Thomas M. Conte. A benchmark characterization of the EEMBC benchmark suite. *IEEE Micro*, 29 (5):18–29, Sep. 2009. DOI: 10.1109/MM.2009.74.

[142] Andrew Putnam, Aaron Smith, and Doug Burger. Dynamic vectorization in the E2 dynamic multicore architecture. *ACM SIGARCH Computer Architecture News*, 38(4):27–32, Jan. 2010. DOI: 10.1145/1926367.1926373.

[143] Qualcomm Inc. Qualcomm Technologies announces world's first commercial 64-bit octa-core chipset with integrated 5 mode global LTE, Feb. 2014. Available: http://www.qualcomm.com/media/releases/2014/02/24/qualcomm-technologies-announces-worlds-first-commercial-64-bit-octa-core. Accessed 2014-02-25.

[144] Wei Quan and Andy D. Pimentel. An iterative multi-application mapping algorithm for heterogeneous MPSoCs. In *Proceedings of the 11th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*. IEEE, Oct. 2013. DOI: 10.1109/ESTIMedia.2013.6704510.

[145] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. Available: http://www.R-project.org/.

[146] Arun Raghavan, Laurel Emurian, Lei Shao, Marios Papaefthymiou, Kevin P. Pipe, Thomas F. Wenisch, and Milo M. K. Martin. Computational sprinting on a hardware/software testbed. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2013. DOI: 10.1145/2451116.2451135.

[147] Ivan Ratković, Oscar Palomar, Milan Stanić, Osman S. Ünsal, Adrian Cristal, and Mateo Valero. On the selection of adder unit in energy efficient vector processing. In *14th International Symposium on Quality Electronic Design (ISQED)*, March 2013. DOI: 10.1109/ISQED.2013.6523602.

[148] Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the 37th International Symposium on Computer Architecture (ISCA)*. ACM, 2010. DOI: 10.1145/1815961.1816002.

[149] Shaolei Ren, Yuxiong He, and Kathryn S. McKinley. A theoretical foundation for scheduling and designing heterogeneous processors for interactive applications. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, 2014. DOI: 10.1007/978-3-662-45174-8_11.

[150] Jude A. Rivers, Gary S. Tyson, Edward S. Davidson, and Todd M. Austin. On high-bandwidth data cache design for multi-issue processors. In *Proceedings of the 30th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 1997. DOI: 10.1109/MICRO.1997.645796.

[151] Andrew Rutherford. *ANOVA and ANCOVA*, chapter Traditional and GLM Approaches to Independent Measures Single Factor ANOVA Designs, pages 17–52. John Wiley & Sons, Inc., 2011. DOI: 10.1002/9781118491683.ch2.

[152] Samsung Electronics Co. Ltd. Samsung Exynos, n.d. Available: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/products5octa_5420.html. Accessed 2014-02-17.

[153] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, and Charles R. Moore. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA)*, Jun. 2003. DOI: 10.1109/ISCA.2003.1207019.

[154] Karthikeyan Sankaralingam, Ramadass Nagarajan, Robert G. McDonald, Rajagopalan Desikan, Saurabh Drolia, M. S. Govindan, Paul Gratz, Divya Gulati, Heather Hanson, Changkyu Kim, Haiming Liu, Nitya Ranganathan, Simha Sethumadhavan, Sadia Sharif, Premkishore Shivakumar, Stephen W. Keckler, and Doug Burger. Distributed microarchitectural protocols in the TRIPS prototype processor. In *Proceedings of the 39th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2006. DOI: 10.1109/MICRO.2006.19.

[155] Vinay Saripalli, Guangyu Sun, Asit Mishra, Yuan Xie, Suman Datta, and Vijaykrishnan Narayanan. Exploiting heterogeneity for energy efficiency in chip multiprocessors. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(2):109–119, Jun. 2011. DOI: 10.1109/JETCAS.2011.2158343.

[156] Serpil Sayın. Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. *Mathematical Programming*, 87(3):543–560, 2000. DOI: 10.1007/s101070050011.

[157] Yiannakis Sazeides, Rakesh Kumar, Dean M. Tullsen, and Theofanis Constantinou. The danger of interval-based power efficiency metrics: When worst is best. *Computer Architecture Letters*, 4(1), Jan. 2005. DOI: 10.1109/L-CA.2005.2.

[158] Yakun Sophia Shao and David Brooks. ISA-independent workload characterization and its implications for specialized architectures. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2013. DOI: 10.1109/ISPASS.2013.6557175.

[159] Timothy Sherwood, Erez Perelman, and Brad Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001. DOI: 10.1109/PACT.2001.953283.

[160] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2002. DOI: 10.1145/605397.605403.

[161] Eran Shifer and Shlomo Weiss. Low-latency adaptive mode transitions and hierarchical power management in asymmetric clustered cores. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(3), Sep. 2008. DOI: 10.1145/2499901.

[162] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture (MICRO)*. ACM, 2009. DOI: 10.1145/1669112.1669135.

[163] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. A design space exploration methodology for application specific MPSoC design. In

*Proceedings of the Computer Society Symposium on VLSI (ISVLSI)*. IEEE, Jul. 2011. DOI: 10.1109/ISVLSI.2011.44.

[164] Aaron Smith, Jon Gibson, Bertrand A. Maher, Nicholas Nethercote, Bill Yoder, Doug Burger, Kathryn S. McKinley, and James H. Burrill. Compiling for EDGE architectures. In *Proceedings of the 4th International Symposium on Code Generation and Optimization (CGO)*. IEEE, Mar. 2006. DOI: 10.1109/CGO.2006.10.

[165] Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000. DOI: 10.1145/378993.379244.

[166] Allan Snavely, Dean M. Tullsen, and Geoff Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM, 2002. DOI: 10.1145/511334.511343.

[167] Tyler Sondag and Hridesh Rajan. Phase-based tuning for better utilization of performance-asymmetric multicore processors. In *Proceedings of the 9th International Symposium on Code Generation and Optimization (CGO)*. IEEE, Apr. 2011. DOI: 10.1109/CGO.2011.5764670.

[168] Vasileios Spiliopoulos, Akash Bagdia, Andreas Hansson, Peter Aldworth, and Stefanos Kaxiras. Introducing DVFS-management in a full-system simulator. In *Proceedings of the 21st International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Aug. 2013. DOI: 10.1109/MASCOTS.2013.75.

[169] Standard Performance Evaluation Corporation. SPEC CPU 2006, Aug. 2015. Available: https://www.spec.org/cpu2006/. Accessed 2016-03-01.

[170] Bo Su, Junli Gu, Li Shen, Wei Huang, Joseph L. Greathouse, and Zhiying Wang. PPEP: Online performance, power, and energy prediction framework and DVFS space exploration. In *Proceedings of the 47th International Symposium on Microarchitecture (MICRO)*. IEEE, Dec. 2014. DOI: 10.1109/MICRO.2014.17.

[171] Karthik T. Sundararajan, Timothy M. Jones, and Nigel P. Topham. The smart cache: An energy-efficient cache architecture through dynamic adaptation. *International Journal of Parallel Programming*, 41(2):305–330, 2013. DOI: 10.1007/s10766-012-0220-y.

[172] Dam Sunwoo, William Wang, Mrinmoy Ghosh, Chander Sudanthi, Geoffrey Blake, Christopher D. Emmons, and Nigel C. Paver. A structured approach to the simulation, analysis and characterization of smartphone applications. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*. IEEE, Sep. 2013. DOI: 10.1109/IISWC.2013.6704677.

[173] Karthik Swaminathan, Emre Kultursay, Vinay Saripalli, Vijaykrishnan Narayanan, Mahmut Kandemir, and Suman Datta. Improving energy efficiency of multi-threaded applications using heterogeneous CMOS-TFET multicores. In *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, Aug. 2011. DOI: 10.1109/ISLPED.2011.5993644.

[174] Andrew D. Targhetta, Donald E. Owen Jr., and Paul V. Gratz. The design space of ultra-low energy asymmetric cryptography. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2014. DOI: 10.1109/ISPASS.2014.6844461.

[175] Michael Bedford Taylor, Walter Lee, Jason E. Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Sungtae Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman P. Amarasinghe, and Anant Agarwal. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In *Proceedings of the 31st International Symposium on Computer Architecture (ISCA)*, Jun. 2004. DOI: 10.1109/ISCA.2004.1310759.

[176] Jürgen Teich. Invasive algorithms and architectures (invasive algorithmen und architekturen). *IT - Information Technology*, 50(5):300–310, 2008. DOI: 10.1524/itit.2008.0499.

[177] Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. Invasive computing: An overview. In Michael Hübner and Jürgen Becker, editors, *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*, pages 241–268. Springer New York, 2011. DOI: 10.1007/978-1-4419-6460-1_11.

[178] Shyamkumar Thoziyoor, Naveen Muralimanohar, and Norman P. Jouppi. CACTI 5.0. Technical report, HP Laboratories, Oct. 2007. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-167.pdf.

[179] Erik Tomusk. Practicalities of design space exploration with gem5 and McPAT. Presentation at the HiPEAC Computing Systems Weak, Oct. 2012. Available: http://old.hipeac.net/system/files/csw-2012a-tomusk.pdf.

[180] Yatish Turakhia, Bharathwaj Raghunathan, Siddharth Garg, and Diana Marculescu. HaDeS: Architectural synthesis for heterogeneous dark silicon chip multi-processors. In *Proceedings of the Design Automation Conference (DAC)*. IEEE, May 2013.

[181] Rafael Ubal, Dana Schaa, Perhaad Mistry, Xiang Gong, Yash Ukidave, Zhongliang Chen, Gunar Schirner, and David R. Kaeli. Exploring the heterogeneous design space for both performance and reliability. In *Proceedings of the 51st Annual Design Automation Conference (DAC)*. ACM, 2014. DOI: 10.1145/2593069.2596680.

[182] Kenzo Van Craeynest and Lieven Eeckhout. Understanding fundamental design choices in single-ISA heterogeneous multicore architectures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4), Jan. 2013. DOI: 10.1145/2400682.2400691.

[183] Sam Van den Steen, Sander De Pestel, Moncef Mechri, Stijn Eyerman, Trevor E. Carlson, David Black-Schaffer, Erik Hagersten, and Lieven Eeckhout. Micro-architecture independent analytical processor performance and power modeling. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2015. DOI: 10.1109/ISPASS.2015.7095782.

[184] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2010. DOI: 10.1145/1736020.1736044.

[185] Ganesh Venkatesh, Jack Sampson, Nathan Goulding-Hotta, Sravanthi Kota Venkata, Michael Bedford Taylor, and Steven Swanson. QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *Proceedings of the 44$^{th}$ International Symposium on Microarchitecture (MICRO)*. ACM, 2011. DOI: 10.1145/2155620.2155640.

[186] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11:2837–2854, Dec. 2010. Available: http://dl.acm.org/citation.cfm?id=1756006.1953024.

[187] Richard Vuduc, James W. Demmel, and Jeff Bilmes. Statistical models for empirical search-based performance tuning. *International Journal of High Performance Computing Applications*, 18(1):65–94, Feb. 2004. DOI: 10.1177/1094342004041293.

[188] Hao Wang, Vijay Sathish, Ripudaman Singh, Michael J. Schulte, and Nam Sung Kim. Workload and power budget partitioning for single-chip heterogeneous processors. In *Proceedings of the 21$^{st}$ International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2012. DOI: 10.1145/2370816.2370873.

[189] Xiaohang Wang, Zhiming Li, Mei Yang, Yingtao Jiang, Masoud Daneshtalab, and Terrence Mak. A low cost, high performance dynamic-programming-based adaptive power allocation scheme for many-core architectures in the dark silicon era. In *Proceedings of the 11$^{th}$ Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*. IEEE, Oct. 2013. DOI: 10.1109/ESTIMedia.2013.6704504.

[190] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. SimFlex: Statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, Jul. 2006. DOI: 10.1109/MM.2006.79.

[191] John Robert Wernsing and Greg Stitt. Elastic computing: A framework for transparent, portable, and adaptive multi-core heterogeneous computing. In *Proceedings of the SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*. ACM, 2010. DOI: 10.1145/1755888.1755906.

[192] Youfeng Wu, Shiliang Hu, Edson Borin, and Cheng Wang. A HW/SW co-designed heterogeneous multi-core virtual machine for energy-efficient general purpose computing. In *Proceedings of the 9th International Symposium on Code Generation and Optimization*. IEEE, Apr. 2011. DOI: 10.1109/CGO.2011.5764691.

[193] Ahmad Yasin. A top-down method for performance analysis and counters architecture. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, March 2014. DOI: 10.1109/ISPASS.2014.6844459.

[194] Wei Zhang, Hang Zhang, and John Lach. Adaptive front-end throttling for superscalar processors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2014. DOI: 10.1145/2627369.2627633.

[195] Hongtao Zhong, Steven A. Lieberman, and Scott A. Mahlke. Extending multicore architectures to exploit hybrid parallelism in single-thread applications. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Feb. 2007. DOI: 10.1109/HPCA.2007.346182.

[196] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4), Nov. 1999. DOI: 10.1109/4235.797969.

[197] Davide Zoni, Simone Corbetta, and William Fornaciari. HANDS: Heterogeneous architectures and networks-on-chip design and simulation. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2012. DOI: 10.1145/2333660.2333721.

[198] Victor Zyuban and Peter M. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Proceedigns of the International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2000. DOI: 10.1109/LPE.2000.155258.

[199] Victor Zyuban and Philip Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In

*Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2002. DOI: 10.1145/566408.566451.